

Package ‘terra’

April 8, 2023

Type Package

Title Spatial Data Analysis

Version 1.7-23

Date 2023-04-06

Depends R (>= 3.5.0)

Suggests parallel, tinytest, ncdf4, sf (>= 0.9-8), deldir, XML,
leaflet, htmlwidgets

LinkingTo Rcpp

Imports methods, Rcpp (>= 1.0-10)

SystemRequirements GDAL (>= 2.2.3), GEOS (>= 3.4.0), PROJ (>= 4.9.3),
sqlite3

Encoding UTF-8

Language en-US

Maintainer Robert J. Hijmans <r.hijmans@gmail.com>

Description Methods for spatial data analysis with vector (points, lines, polygons) and raster (grid) data. Methods for vector data include geometric operations such as intersect and buffer. Raster methods include local, focal, global, zonal and geometric operations. The predict and interpolate methods facilitate the use of regression type (interpolation, machine learning) models for spatial prediction, including with satellite remote sensing data. Processing of very large files is supported. See the manual and tutorials on <<https://rspatial.org/terra/>> to get started. 'terra' is a replacement for the 'raster' package ('terra' can do more, and it is faster and easier to use).

License GPL (>= 3)

URL <https://rspatial.org/>

BugReports <https://github.com/rspatial/terra/issues/>

LazyLoad yes

NeedsCompilation yes

Author Robert J. Hijmans [cre, aut] (<<https://orcid.org/0000-0001-5872-2872>>),
Roger Bivand [ctb] (<<https://orcid.org/0000-0003-2392-6140>>),
Edzer Pebesma [ctb] (<<https://orcid.org/0000-0001-8049-7069>>),
Michael D. Sumner [ctb]

Repository CRAN

Date/Publication 2023-04-08 14:50:02 UTC

R topics documented:

terra-package	7
activeCat	20
add	21
add_legend	22
adjacent	22
aggregate	24
align	25
all.equal	27
animate	28
app	29
approximate	31
Arith-methods	32
as.character	33
as.data.frame	34
as.list	35
as.raster	36
as.spatvector	37
atan2	38
autocorrelation	39
barplot	41
boundaries	42
boxplot	43
buffer	44
c	45
cartogram	46
catalyze	47
cells	48
cellSize	49
centroids	51
clamp	52
clamp_ts	53
classify	54
click	56
coerce	57
colors	58
combineGeoms	59
Compare-methods	61
compareGeom	62
concat	63
contour	64
convHull	65
costDist	66

cover	67
crds	68
crop	69
crosstab	71
crs	72
deepcopy	74
densify	75
density	76
deprecated	77
depth	77
describe	78
diff	79
dimensions	80
direction	82
disagg	83
distance	84
dots	87
draw	88
elongate	89
erase	90
expanse	91
ext	93
extend	94
extract	95
extremes	98
factors	99
fillHoles	101
fillTime	102
flip	103
focal	104
focal3D	106
focalCpp	107
focalMat	109
focalPairs	110
focalReg	112
focalValues	113
freq	113
gaps	114
gdal	115
geom	116
geomtype	117
global	118
graticule	119
gridDistance	120
halo	121
headtail	122
hist	123
ifel	124

image	125
impose	126
initialize	127
inplace	128
inset	130
interpIDW	131
interpNear	133
interpolation	134
intersect	137
is.bool	139
is.lonlat	140
lapp	141
layerCor	144
linearUnits	145
lines	146
makeTiles	148
makeVRT	149
map.pal	150
mask	151
match	153
Math-methods	154
mem	155
merge	156
mergeTime	157
meta	158
modal	159
mosaic	160
na.omit	161
NAflag	162
names	163
nearest	165
normalize.longitude	166
north	166
not.na	168
options	169
origin	170
pairs	171
panel	172
patches	173
perim	174
persp	175
plet	176
plot	178
plotRGB	182
plot_extent	184
plot_graticule	185
predict	186
project	189

quantile	192
query	193
rangeFill	194
rapp	195
rast	197
rasterize	200
rasterizeGeom	201
rasterizeWin	203
rcl	204
readwrite	205
rectify	207
relate	208
rep	210
replace_dollar	211
replace_layers	212
replace_values	213
resample	214
rescale	215
RGB	216
roll	217
rotate	219
same.crs	220
sapp	221
sbar	222
scale	223
scatterplot	224
scoff	225
sds	226
segregate	227
sel	228
selectHighest	229
selectRange	230
serialize	231
setValues	232
shade	233
sharedPaths	234
shift	235
sieve	236
simplifyGeom	237
sort	238
sources	239
SpatExtent-class	240
SpatRaster-class	241
spatSample	241
SpatVector-class	243
spin	244
split	245
sprc	246

stretch	247
subset	248
subset_dollar	249
subset_double	251
subset_single	252
subst	253
summarize	254
summary	257
svc	258
symdif	259
tapp	260
terrain	261
text	263
tighten	264
time	265
tmpFiles	266
topology	267
transpose	268
trim	269
union	270
unique	271
units	272
update	273
valid	274
values	275
vect	276
vector_layers	279
viewshed	280
voronoi	281
vrt	282
weighted.mean	283
where	284
which.lyr	285
width	286
window	287
wrap	288
writeCDF	289
writeRaster	290
writeVector	292
xmin	293
xyRowColCell	294
zonal	297
zoom	299

terra-package	<i>Description of the methods in the terra package</i>
---------------	--

Description

terra provides methods to manipulate geographic (spatial) data in "raster" and "vector" form. Raster data divide space into rectangular grid cells and they are commonly used to represent spatially continuous phenomena, such as elevation or the weather. Satellite images also have this data structure, and in that context grid cells are often referred to as pixels. In contrast, "vector" spatial data (points, lines, polygons) are typically used to represent discrete spatial entities, such as a road, country, or bus stop.

The package implements two main classes (data types): SpatRaster and SpatVector. SpatRaster supports handling large raster files that cannot be loaded into memory; local, focal, zonal, and global raster operations; polygon, line and point to raster conversion; integration with modeling methods to make spatial predictions; and more. SpatVector supports all types of geometric operations such as intersections.

Additional classes include SpatExtent, which is used to define a spatial extent (bounding box); SpatRasterDataset, which represents a collection of sub-datasets for the same area. Each sub-dataset is a SpatRaster with possibly many layers, and may, for example, represent different weather variables; and SpatRasterCollection and SpatVectorCollection that are equivalent to lists of SpatRaster or SpatVector objects. There is also a SpatGraticule class to assist in adding a longitude/latitude lines and labels to a map with another coordinate reference system.

These classes hold a C++ pointer to the data "reference class" and that creates some limitations. They cannot be recovered from a saved R session either or directly passed to nodes on a computer cluster. Generally, you should use [writeRaster](#) to save SpatRaster objects to disk (and pass a filename or cell values to cluster nodes). Also see [wrap](#).

The terra package is conceived as a replacement of the raster package. terra has a very similar, but simpler, interface, and it is faster than raster. At the bottom of this page there is a table that shows differences in the methods between the two packages.

Below is a list of some of the most important methods grouped by theme.

SpatRaster

I. Creating, combining and sub-setting

rast	Create a SpatRaster from scratch, file, or another object
c	Combine SpatRasters (multiple layers)
add<-	Add a SpatRaster to another one
subset or [[, or \$	Select layers of a SpatRaster
selectRange	Select cell values from different layers using an index layer

II. Changing the spatial extent or resolution

Also see the methods in section VIII

<code>merge</code>	Combine SpatRasters with different extents (but same origin and resolution)
<code>mosaic</code>	Combine SpatRasters with different extents using a function for overlapping cells
<code>crop</code>	Select a geographic subset of a SpatRaster
<code>extend</code>	Add rows and/or columns to a SpatRaster
<code>trim</code>	Trim a SpatRaster by removing exterior rows and/or columns that only have NAs
<code>aggregate</code>	Combine cells of a SpatRaster to create larger cells
<code>disagg</code>	Subdivide cells
<code>resample</code>	Resample (warp) values to a SpatRaster with a different origin and/or resolution
<code>project</code>	Project (warp) values to a SpatRaster with a different coordinate reference system
<code>shift</code>	Adjust the location of SpatRaster
<code>flip</code>	Flip values horizontally or vertically
<code>rotate</code>	Rotate values around the date-line (for lon/lat data)
<code>t</code>	Transpose a SpatRaster

III. Local (cell based) methods

Apply-like methods:

<code>app</code>	Apply a function to all cells, across layers, typically to summarize (as in <code>base::apply</code>)
<code>tapp</code>	Apply a function to groups of layers (as in <code>base::tapply</code> and <code>stats::aggregate</code>)
<code>lapp</code>	Apply a function to using the layers of a SpatRaster as variables
<code>sapp</code>	Apply a function to each layer
<code>rapp</code>	Apply a function to a spatially variable range of layers

Arithmetic, logical, and standard math methods:

<code>Arith-methods</code>	Standard arithmetic methods (+, -, *, ^, %, %/, /)
<code>Compare-methods</code>	Comparison methods for SpatRaster (==, !=, >, <, <=, >=, <code>is.na</code> , <code>is.finite</code>)
<code>not.na</code>	a one-step equivalent to <code>!is.na</code>
<code>Summary-methods</code>	<code>mean</code> , <code>max</code> , <code>min</code> , <code>median</code> , <code>sum</code> , <code>range</code> , <code>prod</code> , <code>any</code> , <code>all</code> , <code>stdev</code> , <code>which.min</code> , <code>which.max</code> , <code>anyNA</code> , <code>noNA</code> , <code>allNA</code>
<code>Logic-methods</code>	Boolean methods (!, &,)
<code>Math-methods</code>	<code>abs</code> , <code>sign</code> , <code>sqrt</code> , <code>ceiling</code> , <code>floor</code> , <code>trunc</code> , <code>cummax</code> , <code>cummin</code> , <code>cumprod</code> , <code>cumsum</code> , <code>log</code> , <code>log10</code> , <code>log2</code> , <code>log1p</code> , <code>acos</code> , <code>acosh</code> , <code>asin</code> , <code>asinh</code> , <code>atan</code> , <code>atanh</code> , <code>exp</code> , <code>expm1</code> , <code>cos</code> , <code>cosh</code> , <code>sin</code> , <code>sinh</code> , <code>tan</code> , <code>tanh</code> , <code>round</code> , <code>signif</code>
<code>as.bool</code>	create a Boolean (logical) SpatRaster
<code>as.int</code>	create an integer (whole numbers) SpatRaster

Other methods:

<code>approximate</code>	Compute missing values for cells by interpolation across layers
<code>roll</code>	Rolling functions such as the rolling mean
<code>cellSize</code>	Compute the area of cells
<code>classify</code>	(Re-)classify values
<code>subst</code>	Substitute (replace) cell values
<code>cover</code>	First layer covers second layer except where the first layer is NA
<code>init</code>	Initialize cells with new values
<code>mask</code>	Replace values in a SpatRaster based on values in another SpatRaster
<code>which.lyr</code>	which is the first layer that is TRUE?
<code>segregate</code>	Make a 0/1 layer for each unique value
<code>rangeFill</code>	Make a 0/1 SpatRaster for a time series

IV. Zonal and global methods

<code>expanse</code>	Compute the summed area of cells
<code>crosstab</code>	Cross-tabulate two SpatRasters
<code>freq</code>	Frequency table of SpatRaster cell values
<code>global</code>	Summarize SpatRaster cell values with a function
<code>quantile</code>	Quantiles
<code>layerCor</code>	Correlation between layers
<code>stretch</code>	Stretch values
<code>scale</code>	Scale values
<code>summary</code>	Summary of the values of a SpatRaster (quantiles and mean)
<code>unique</code>	Get the unique values in a SpatRaster
<code>zonal</code>	Summarize a SpatRaster by zones in another SpatRaster

V. Situation (spatial context) based methods

<code>adjacent</code>	Identify cells that are adjacent to a set of cells of a SpatRaster
<code>boundaries</code>	Detection of boundaries (edges)
<code>distance</code>	Shortest distance to a cell that is not NA or to or from a vector object
<code>gridDist</code>	Shortest distance through adjacent grid cells
<code>costDist</code>	Shortest distance considering cell-varying friction
<code>direction</code>	Direction (azimuth) to or from cells that are not NA
<code>focal</code>	Focal (neighborhood; moving window) functions
<code>focal3D</code>	Three dimensional (row, col, lyr) focal functions
<code>focalCpp</code>	Faster focal by using custom C++ functions
<code>focalReg</code>	Regression between layers for focal areas
<code>focalPairs</code>	Apply a function (e.g. a correlation coefficient) to focal values for pairs of layers
<code>patches</code>	Find patches (clumps)

<code>sieve</code>	Sieve filter to remove small patches
<code>terrain</code>	Compute slope, aspect and other terrain characteristics from elevation data
<code>viewshed</code>	Compute viewshed (showing areas that are visible from a particular location)
<code>shade</code>	Compute hill shade from slope and aspect layers
<code>autocor</code>	Compute global or local spatial autocorrelation

VI. Model predictions

<code>predict</code>	Predict a non-spatial model to a SpatRaster
<code>interpolate</code>	Predict a spatial model to a SpatRaster
<code>interpIDW</code>	Inverse-distance-weighted interpolation
<code>interpNear</code>	Nearst neighbor interpolation

VII. Accessing cell values

Apart from the function listed below, you can also use indexing with `[]` with cell numbers, and row and/or column numbers

<code>values</code>	cell values (fails with very large rasters)
<code>values<-</code>	Set new values to the cells of a SpatRaster
<code>setValues</code>	Set new values to the cells of a SpatRaster
<code>as.matrix</code>	Get cell values as a matrix
<code>as.array</code>	Get cell values as an array
<code>extract</code>	Extract cell values from a SpatRaster (e.g., by cell, coordinates, polygon)
<code>spatSample</code>	Regular or random sample
<code>minmax</code>	Get the minimum and maximum value of the cells of a SpatRaster (if known)
<code>setMinMax</code>	Compute the minimum and maximum value of a SpatRaster if these are not known
<code>extract</code>	spatial queries of a SpatRaster with a SpatVector

VIII. Getting and setting dimensions

Get or set basic parameters of SpatRasters. If there are values associated with a SpatRaster object (either in memory or via a link to a file) these are lost when you change the number of columns or rows or the resolution. This is not the case when the extent is changed (as the number of columns and rows will not be affected). Similarly, with `crs` you can set the coordinate reference system, but this does not transform the data (see [project](#) for that).

<code>ncol</code>	The number of columns
<code>nrow</code>	The number of rows
<code>ncell</code>	The number of cells (can not be set directly, only via <code>ncol</code> or <code>nrow</code>)
<code>res</code>	The resolution (x and y)
<code>nlyr</code>	Get or set the number of layers
<code>names</code>	Get or set the layer names
<code>xres</code>	The x resolution (can be set with <code>res</code>)
<code>yres</code>	The y resolution (can be set with <code>res</code>)
<code>xmin</code>	The minimum x coordinate (or longitude)
<code>xmax</code>	The maximum x coordinate (or longitude)
<code>ymin</code>	The minimum y coordinate (or latitude)
<code>ymax</code>	The maximum y coordinate (or latitude)
<code>ext</code>	Get or set the extent (minimum and maximum x and y coordinates ("bounding box"))
<code>origin</code>	The origin of a <code>SpatRaster</code>
<code>crs</code>	The coordinate reference system (map projection)
<code>is.lonlat</code>	Test if an object has (or may have) a longitude/latitude coordinate reference system
<code>sources</code>	Get the filename(s) to which a <code>SpatRaster</code> is linked
<code>inMemory</code>	Are the data sources in memory (or on disk)?
<code>compareGeom</code>	Compare the geometry of <code>SpatRasters</code>
<code>NAflag</code>	Set the NA value (for reading from a file with insufficient metadata)

IX. Computing row, column, cell numbers and coordinates

Cell numbers start at 1 in the upper-left corner. They increase within rows, from left to right, and then row by row from top to bottom. Likewise, row numbers start at 1 at the top of the raster, and column numbers start at 1 at the left side of the raster.

<code>xFromCol</code>	x-coordinates from column numbers
<code>yFromRow</code>	y-coordinates from row numbers
<code>xFromCell</code>	x-coordinates from row numbers
<code>yFromCell</code>	y-coordinates from cell numbers
<code>xyFromCell</code>	x and y coordinates from cell numbers
<code>colFromX</code>	Column numbers from x-coordinates (or longitude)
<code>rowFromY</code>	Row numbers from y-coordinates (or latitude)
<code>rowColFromCell</code>	Row and column numbers from cell numbers
<code>cellFromXY</code>	Cell numbers from x and y coordinates
<code>cellFromRowCol</code>	Cell numbers from row and column numbers
<code>cellFromRowColCombine</code>	Cell numbers from all combinations of row and column numbers
<code>cells</code>	Cell numbers from an <code>SpatVector</code> or <code>SpatExtent</code>

X. Time related methods

<code>time</code>	Get or set time
<code>fillTime</code>	can add empty layers in between existing layers to assure that the time step between layers is constant
<code>mergeTime</code>	combine multiple rasters, perhaps partly overlapping in time, into a single time series

XI. Methods for categorical rasters

<code>is.factor</code>	Are there categorical layers?
<code>levels</code>	Get active categories, or set categories
<code>activeCat</code>	Get or set the active category
<code>cats</code>	Get categories (active and inactive)
<code>set.cats</code>	Set categories in place
<code>concat</code>	Combine SpatRasters with different categories
<code>catalyze</code>	Create a layer for each category
<code>as.numeric</code>	use the active category to create a non-categorical SpatRaster
<code>as.factor</code>	Make the layers of a SpatRaster categorical

XII. Writing SpatRaster files

Basic:

<code>writeRaster</code>	Write all values of SpatRaster to disk. You can set the filetype, datatype, compression.
<code>writeCDF</code>	Write SpatRaster data to a netCDF file

Advanced:

<code>readStart</code>	Open file connections for efficient multi-chunk reading
<code>readStop</code>	Close file connections
<code>writeStart</code>	Open a file for writing
<code>writeValues</code>	Write some values
<code>writeStop</code>	Close the file after writing
<code>blocks</code>	Get blocksize for reading files (when not writing)

XIII. Miscellaneous SpatRaster methods

<code>terraOptions</code>	Show, set, or get session options, mostly to control memory use and to set write options
<code>sources</code>	Show the data sources of a SpatRaster
<code>tmpFiles</code>	Show or remove temporary files

<code>mem_info</code>	memory needs and availability
<code>inMemory</code>	Are the cell values in memory?

XIV. SpatRasterDataset

A `SpatRasterDataset` contains `SpatRaster` objects that are sub-datasets for the same area. They all have the same extent and resolution.

<code>sds</code>	Create a <code>SpatRasterDataset</code> from a file with subdatasets (ncdf or hdf) or from <code>SpatRaster</code> objects
<code>[</code> or <code>\$</code>	Extract a <code>SpatRaster</code>
<code>names</code>	Get the names of the sub-datasets

XV. SpatRasterCollections

A `SpatRasterCollection` is a vector of `SpatRaster` objects. Unlike for a `SpatRasterDataset`, there the extent and resolution of the `SpatRasters` do not need to match each other.

<code>sprc</code>	create a <code>SpatRasterCollection</code> from (a list of) <code>SpatRaster</code> objects
<code>length</code>	how many <code>SpatRasters</code> does the <code>SpatRasterCollection</code> have?
<code>crop</code>	crop a <code>SpatRasterCollection</code>
<code>impose</code>	force the members of <code>SpatRasterCollection</code> to the same geometry
<code>merge</code>	merge the members of a <code>SpatRasterCollection</code>
<code>mosaic</code>	mosaic (merge with a function for overlapping areas) the members of a <code>SpatRasterCollection</code>
<code>[</code>	extract a <code>SpatRaster</code>

SpatVector

XVI. Create SpatVector objects

<code>vect</code>	Create a <code>SpatVector</code> from a file (for example a "shapefile") or from another object
<code>vector_layers</code>	list or delete layers in a vector database such as GPKG
<code>rbind</code>	append <code>SpatVectors</code> of the same geometry type
<code>unique</code>	remove duplicates
<code>na.omit</code>	remove empty geometries and/or fields that are NA
<code>project</code>	Project a <code>SpatVector</code> to a different coordinate reference system
<code>writeVector</code>	Write <code>SpatVector</code> data to disk
<code>centroids</code>	Get the centroids of a <code>SpatVector</code>
<code>voronoi</code>	Voronoi diagram

<code>delaunay</code>	Delaunay triangles
<code>convHull</code>	Compute the convex hull of a <code>SpatVector</code>
<code>fillHoles</code>	Remove or extract holes from polygons

XVII. Properties of `SpatVector` objects

<code>geom</code>	returns the geometries as matrix or WKT
<code>crds</code>	returns the coordinates as a matrix
<code>linearUnits</code>	returns the linear units of the crs (in meter)
<code>ncol</code>	The number of columns (of the attributes)
<code>nrow</code>	The number of rows (of the geometries and attributes)
<code>names</code>	Get or set the layer names
<code>ext</code>	Get the extent (minimum and maximum x and y coordinates ("bounding box"))
<code>crs</code>	The coordinate reference system (map projection)
<code>is.lonlat</code>	Test if an object has (or may have) a longitude/latitude coordinate reference system

XVIII. Geometric queries

<code>adjacent</code>	find adjacent polygons
<code>expanse</code>	computes the area covered by polygons
<code>nearby</code>	find nearby geometries
<code>nearest</code>	find the nearest geometries
<code>relate</code>	geometric relationships such as "intersects", "overlaps", and "touches"
<code>perim</code>	computes the length of the perimeter of polygons, and the length of lines

XIX. Geometric operations

<code>erase</code> or <code>"-"</code>	erase (parts of) geometries
<code>intersect</code> or <code>"*"</code>	intersect geometries
<code>union</code> or <code>"+"</code>	Merge geometries
<code>cover</code>	update polygons
<code>symdif</code>	symmetrical difference of two polygons
<code>aggregate</code>	dissolve smaller polygons into larger ones
<code>buffer</code>	buffer geometries

<code>disagg</code>	split multi-geometries into separate geometries
<code>crop</code>	clip geometries using a rectangle (<code>SpatExtent</code>) or <code>SpatVector</code>

XX. SpatVector attributes

We use the term "attributes" for the tabular data (`data.frame`) associated with vector geometries.

<code>extract</code>	spatial queries between <code>SpatVector</code> and <code>SpatVector</code> (e.g. point in polygons)
<code>sel</code>	select - interactively select geometries
<code>click</code>	identify attributes by clicking on a map
<code>merge</code>	Join a table with a <code>SpatVector</code>
<code>as.data.frame</code>	get attributes as a <code>data.frame</code>
<code>as.list</code>	get attributes as a list
<code>values</code>	Get the attributes of a <code>SpatVector</code>
<code>values<-</code>	Set new attributes to the geometries of a <code>SpatRaster</code>
<code>sort</code>	sort <code>SpatVector</code> by the values in a field

XXI. Change geometries (for display, experimentation)

<code>shift</code>	change the position geometries by shifting their coordinates in horizontal and/or vertical direction
<code>spin</code>	rotate geometries around an origin
<code>rescale</code>	shrink (or expand) geometries, for example to make an inset map
<code>flip</code>	flip geometries vertically or horizontally
<code>t</code>	transpose geometries (switch x and y)

XXII. Geometry properties and topology

<code>width</code>	the minimum diameter of the geometries
<code>clearance</code>	the minimum clearance of the geometries
<code>sharedPaths</code>	shared paths (arcs) between line or polygon geometries
<code>simplifyGeom</code>	simplify geometries
<code>gaps</code>	find gaps between polygon geometries
<code>fillHoles</code>	get or remove the polygon holes
<code>makeNodes</code>	create nodes on lines
<code>mergeLines</code>	connect lines to form polygons
<code>removeDupNodes</code>	remove duplicate nodes in geometries and optionally rounds the coordinates
<code>is.valid</code>	check if geometries are valid

<code>makeValid</code>	attempt to repair invalid geometries
<code>snap</code>	make boundaries of geometries identical if they are very close to each other
<code>erase</code> (single argument)	remove parts of geometries that overlap
<code>union</code> (single argument)	create new polygons such that there are no overlapping polygons
<code>rotate</code>	rotate to (dis-) connect them across the date-line
<code>normalize.longitude</code>	move geomtries that are outside of the -180 to 180 degrees range.
<code>elongate</code>	make lines longer by extending both sides
<code>combineGeoms</code>	combine geometries that overlap, share a border, or are within a minimum distance of each other

XXIII. SpatVectorCollections

A SpatVectorCollection is a vector of SpatVector objects.

<code>svc</code>	create a SpatVectorCollection from (a list of) SpatVector objects
<code>length</code>	how many SpatRasters does the SpatRasterCollection have?
<code>[</code>	extract a SpatVector

Other classes

XXIV. SpatExtent

<code>ext</code>	Create a SpatExtent object. For example to <code>crop</code> a Spatial dataset
<code>intersect</code>	Intersect two SpatExtent objects, same as -
<code>union</code>	Combine two SpatExtent objects, same as +
<code>Math-methods</code>	round/floor/ceiling of a SpatExtent
<code>align</code>	Align a SpatExtent with a SpatRaster
<code>draw</code>	Create a SpatExtent by drawing it on top of a map (plot)

XXV. SpatGraticule

<code>graticule</code>	Create a graticule
<code>crop</code>	crop a graticule
<code>plot<SpatGraticule></code>	plot a graticule

General methods

XXVI. Conversion between spatial data objects from different packages

You can coerce SpatRasters to Raster* objects, after loading the raster package, with `as(object, "Raster")`, or `raster(object)` or `brick(object)` or `stack(object)`

<code>rast</code>	SpatRaster from matrix and other objects
<code>vect</code>	SpatVector from sf or Spatial* vector data
<code>sf::st_as_sf</code>	sf object from SpatVector
<code>rasterize</code>	Rasterizing points, lines or polygons
<code>rasterizeWin</code>	Rasterize points with a moving window
<code>rasterizeGeom</code>	Rasterize attributes of geometries such as "count", "area", or "length"
<code>as.points</code>	Create points from a SpatRaster or SpatVector
<code>as.lines</code>	Create points from a SpatRaster or SpatVector
<code>as.polygons</code>	Create polygons from a SpatRaster
<code>as.contour</code>	Contour lines from a SpatRaster

XXVII. Plotting

Maps:

<code>plot</code>	Plot a SpatRaster or SpatVector. The main method to create a map
<code>panel</code>	Combine multiple plots
<code>points</code>	Add points to a map
<code>lines</code>	Add lines to a map
<code>polys</code>	Add polygons to a map
<code>text</code>	Add text (such as the values of a SpatRaster or SpatVector) to a map
<code>halo</code>	Add text with a halo to a map
<code>image</code>	Alternative to plot to make a map with a SpatRaster
<code>plotRGB</code>	Combine three layers (red, green, blue channels) into a single "real color" plot
<code>plot<SpatGraticule></code>	plot a graticule
<code>sbar</code>	Add a scale bar to a map
<code>north</code>	Add a north arrow to a map
<code>inset</code>	Add a small inset (overview) map
<code>dots</code>	Make a dot-density map
<code>cartogram</code>	Make a cartogram
<code>persp</code>	Perspective plot of a SpatRaster
<code>contour</code>	Contour plot or filled-contour plot of a SpatRaster
<code>colorize</code>	Combine three layers (red, green, blue channels) into a single layer with a color-table

Interacting with a map:

<code>zoom</code>	Zoom in to a part of a map by drawing a bounding box on it
<code>click</code>	Query values of SpatRaster or SpatVector by clicking on a map
<code>sel</code>	Select a spatial subset of a SpatRaster or SpatVector by drawing on a map
<code>draw</code>	Create a SpatExtent or SpatVector by drawing on a map

Other plots:

<code>plot</code>	x-y scatter plot of the values of (a sample of) the layers of two SpatRaster objects
<code>hist</code>	Histogram of SpatRaster values
<code>barplot</code>	Bar plot of a SpatRaster
<code>density</code>	Density plot of SpatRaster values
<code>pairs</code>	Pairs plot for layers in a SpatRaster
<code>boxplot</code>	Box plot of the values of a SpatRaster

Comparison with the raster package**XXVIII. New method names**

terra has a single class SpatRaster for which raster has three (RasterLayer, RasterStack, RasterBrick). Likewise there is a single class for vector data SpatVector that replaces six Spatial* classes. Most method names are the same, but note the following important differences in methods names with the raster package

raster package	terra package
<code>raster, brick, stack</code>	<code>rast</code>
<code>rasterFromXYZ</code>	<code>rast(, type="xyz")</code>
<code>stack, addLayer</code>	<code>c</code>
<code>addLayer</code>	<code>add<-</code>
<code>area</code>	<code>cellSize</code> or <code>expanses</code>
<code>approxNA</code>	<code>approximate</code>
<code>calc</code>	<code>app</code>
<code>cellFromLine, cellFromPolygon,</code>	<code>cells</code>
<code>cellsFromExtent</code>	<code>cells</code>
<code>cellStats</code>	<code>global</code>
<code>clump</code>	<code>patches</code>
<code>compareRaster</code>	<code>compareGeom</code>
<code>corLocal</code>	<code>focalCor</code>
<code>coordinates</code>	<code>crds</code>
<code>couldBeLonLat</code>	<code>is.lonlat</code>
<code>disaggregate</code>	<code>disagg</code>
<code>distanceFromPoints</code>	<code>distance</code>

drawExtent, drawPoly, drawLine	draw
dropLayer	subset
extent	ext
getValues	values
isLonLat, isGlobalLonLat	is.lonlat
layerize	segregate
layerStats	layerCor
movingFun	roll
NAvalue	NAflag
nlayers	nlyr
overlay	lapp
unstack	as.list
projectRaster	project
rasterToPoints	as.points
rasterToPolygons	as.polygons
reclassify, subs, cut	classify
sampleRandom, sampleRegular	spatSample
shapefile	vect
stackApply	tapp
stackSelect	selectRange

XXIX. Changed behavior

Also note that even if function names are the same in terra and raster, their output can be different. In most cases this was done to get more consistency in the returned values (and thus fewer errors in the downstream code that uses them). In other cases it simply seemed better. Here are some examples:

as.polygons	By default, terra returns dissolved polygons
quantile	computes by cell, across layers instead of the other way around
extract	By default, terra returns a matrix, with the first column the sequential ID of the vectors. raster returns a list (for lines or polygons) or a matrix (for points, but without the ID column. You can use <code>list=TRUE</code> to get the results as a list
values	terra always returns a matrix. raster returns a vector for a RasterLayer
Summary-methods	With raster, <code>mean(x, y)</code> and <code>mean(stack(x, y))</code> return the same result, a single layer with the mean of all cell values. This is also what terra returns with <code>mean(c(x, y))</code> , but with <code>mean(x, y)</code> the parallel mean is returned – that is, the computation is done layer-wise, and the number of layers in the output is the same as that of x and y (or the larger of the two if they are not the same). This affects all summary functions (<code>sum</code> , <code>mean</code> , <code>median</code> , <code>which.min</code> , <code>which.max</code> , <code>min</code> , <code>max</code> , <code>prod</code> , <code>any</code> , <code>all</code> , <code>stdev</code>), except <code>range</code> , which is not implemented for this case (you can use <code>min</code> and <code>max</code> instead)

Authors

Except where indicated otherwise, the methods and functions in this package were written by Robert Hijmans. The configuration scripts were written by Roger Bivand. Some of code using the GEOS library was adapted from code by Edzer Pebesma for *sf*. Michael Sumner contributed various bits and pieces.

Acknowledgments

This package is an attempt to climb on the shoulders of giants (GDAL, PROJ, GEOS, NCDF, GeographicLib, Rcpp, R). Many people have contributed by asking questions or **raising issues**. Feedback and suggestions by Márcia Barbosa, Kendon Bell, Andrew Gene Brown, Jean-Luc Dupouey, Krzysztof Dyba, Alex Ilich, Gerald Nelson, Jakub Nowosad and Monika Tomaszewska have been especially helpful.

activeCat

Active category

Description

Get or set the active category of a multi-categorical SpatRaster layer

Usage

```
## S4 method for signature 'SpatRaster'
activeCat(x, layer=1)
## S4 replacement method for signature 'SpatRaster'
activeCat(x, layer=1)<-value
```

Arguments

x	SpatRaster
layer	positive integer, the layer number or name
value	positive integer or character, indicating which column in the categories to use. Note that when a number is used this index is zero based, and "1" refers to the second column. This is because the first column of the categories has the cell values, not categorical labels

Value

integer

See Also

[levels](#), [cats](#)

Examples

```

set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE) + 10
d <- data.frame(id=11:13, cover=c("forest", "water", "urban"), letters=letters[1:3], value=10:12)
levels(r) <- d

activeCat(r)
activeCat(r) <- 3
activeCat(r)

```

add

*Add (in place) a SpatRaster to another SpatRaster object***Description**

Add (in place) a SpatRaster to another SpatRaster object. Comparable with [c](#), but without copying the object.

Usage

```

## S4 replacement method for signature 'SpatRaster,SpatRaster'
add(x)<-value

```

Arguments

x	SpatRaster
value	SpatRaster

Value

SpatRaster

See Also

[c](#)

Examples

```

r <- rast(nrows=5, ncols=9, vals=1:45)
x <- c(r, r*2)
add(x) <- r*3
x

```

add_legend	<i>add a custom legend</i>
------------	----------------------------

Description

Wrapper around [legend](#) that allows adding a custom legend to a map using a keyword such as "topleft" or "bottomright". This function will place the legend in the locations within the mapped area as delineated by the axes.

Usage

```
add_legend(x, y, ...)
```

Arguments

x	The keyword to be used to position the legend (or the x coordinate)
y	The y coordinate to be used to position the legend (is x is also a coordinate)
...	arguments passed to link{legend}

Examples

```
v <- vect(system.file("ex/lux.shp", package="terra"))
plot(v)
points(centroids(v), col="red")
legend("topleft", legend = "centroids", pch = 20, xpd=NA, bg="white", col="red")
add_legend("topright", legend = "centroids", pch = 20, col="red")
```

adjacent	<i>Adjacent cells</i>
----------	-----------------------

Description

Identify cells that are adjacent to a set of raster cells. Or identify adjacent polygons

Usage

```
## S4 method for signature 'SpatRaster'
adjacent(x, cells, directions="rook", pairs=FALSE, include=FALSE)

## S4 method for signature 'SpatVector'
adjacent(x, type="rook", pairs=TRUE, symmetrical=FALSE)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>cells</code>	vector of cell numbers for which adjacent cells should be found. Cell numbers start with 1 in the upper-left corner and increase from left to right and from top to bottom
<code>directions</code>	character or matrix to indicated the directions in which cells are considered connected. The following character values are allowed: "rook" or "4" for the horizontal and vertical neighbors; "bishop" to get the diagonal neighbors; "queen" or "8" to get the vertical, horizontal and diagonal neighbors; or "16" for knight and one-cell queen move neighbors. If <code>directions</code> is a matrix it should have odd dimensions and have logical (or 0, 1) values
<code>pairs</code>	logical. If TRUE, a two-column matrix of pairs of adjacent cells is returned. If <code>x</code> is a <code>SpatRaster</code> and <code>pairs</code> is FALSE, an <code>n*m</code> matrix is returned where the number of rows <code>n</code> is <code>length(cells)</code> and the number of columns <code>m</code> is the number of neighbors requested with <code>directions</code>
<code>include</code>	logical. Should the focal cells be included in the result?
<code>type</code>	character. One of "rook", "queen", "touches", or "intersects". "queen" and "touches" are synonyms. "rook" exclude polygons that touch at a single node only. "intersects" includes polygons that touch or overlap
<code>symmetrical</code>	logical. If TRUE, an adjacent pair is only included once. For example, if polygon 1 is adjacent to polygon 3, the implied adjacency between 3 and 1 is not reported

Value

matrix

See Also

[relate](#), [nearby](#)

Examples

```
r <- rast(nrows=10, ncols=10)
adjacent(r, cells=c(1, 5, 55), directions="queen")
r <- rast(nrows=10, ncols=10, crs="+proj=utm +zone=1 +datum=WGS84")
adjacent(r, cells=11, directions="rook")

#same as
rk <- matrix(c(0,1,0,1,0,1,0,1,0), 3, 3)
adjacent(r, cells=11, directions=rk)

## note that with global lat/lon data the E and W connect
r <- rast(nrows=10, ncols=10, crs="+proj=longlat +datum=WGS84")
adjacent(r, cells=11, directions="rook")

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
a <- adjacent(v, symmetrical=TRUE)
head(a)
```

 aggregate

Aggregate raster or vector data

Description

Aggregate a `SpatRaster` to create a new `SpatRaster` with a lower resolution (larger cells). Aggregation groups rectangular areas to create larger cells. The value for the resulting cells is computed with a user-specified function.

You can also aggregate ("dissolve") a `SpatVector`. This either combines all geometries into one geometry, or it combines the geometries that have the same value for the variable(s) specified with argument `by`.

Usage

```
## S4 method for signature 'SpatRaster'
aggregate(x, fact=2, fun="mean", ..., cores=1, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatVector'
aggregate(x, by=NULL, dissolve=TRUE, fun="mean", count=TRUE, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>fact</code>	positive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers)
<code>fun</code>	function used to aggregate values. Either an actual function, or for the following, their name: "mean", "max", "min", "median", "sum", "modal", "any", "all", "prod", "which.min", "which.max", "sd" (sample standard deviation) and "std" (population standard deviation)
<code>...</code>	additional arguments passed to <code>fun</code> , such as <code>na.rm=TRUE</code>
<code>cores</code>	positive integer. If <code>cores > 1</code> , a 'parallel' package cluster with that many cores is created. Ignored for C++ level implemented functions that are listed under <code>fun</code>
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If <code>TRUE</code> , <code>filename</code> is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster
<code>by</code>	character. The variable(s) used to group the geometries
<code>dissolve</code>	logical. Should borders between aggregated geometries be dissolved?
<code>count</code>	logical. If <code>TRUE</code> and <code>by</code> is not <code>NULL</code> , a variable "agg_n" is included that shows the number of input geometries for each output geometry

Details

Aggregation starts at the upper-left end of a `SpatRaster`. If a division of the number of columns or rows with `factor` does not return an integer, the extent of the resulting `SpatRaster` will be somewhat larger than that of the original `SpatRaster`. For example, if an input `SpatRaster` has 100 columns, and `fact=12`, the output `SpatRaster` will have 9 columns and the maximum x coordinate of the output `SpatRaster` is also adjusted.

The function `fun` should take multiple numbers, and return a single number. For example `mean`, `modal`, `min` or `max`.

It should also accept a `na.rm` argument (or ignore it as one of the 'dots' arguments).

Value

`SpatRaster`

See Also

[disagg](#) to disaggregate

Examples

```
r <- rast()
# aggregated SpatRaster, no values
ra <- aggregate(r, fact=10)

values(r) <- runif(ncell(r))
# aggregated raster, max of the values
ra <- aggregate(r, fact=10, fun=max)

# multiple layers
s <- c(r, r*2)
x <- aggregate(s, 20)

## SpatVector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
va <- aggregate(v, "ID_1")

plot(va, "NAME_1", lwd=5, plg=list(x="topright"), mar=rep(2,4))
lines(v, lwd=3, col="light gray")
lines(va)
text(v, "ID_1", halo=TRUE)
```

Description

Align an SpatExtent with a SpatRaster This can be useful to create a new SpatRaster with the same origin and resolution as an existing SpatRaster. Do not use this to force data to match that really does not match (use e.g. [resample](#) or (dis)aggregate for this).

It is also possible to align a SpatExtent to a clean divisor.

Usage

```
## S4 method for signature 'SpatExtent,SpatRaster'
align(x, y, snap="near")

## S4 method for signature 'SpatExtent,numeric'
align(x, y)
```

Arguments

x	SpatExtent
y	SpatRaster or numeric
snap	Character. One of "near", "in", or "out", to determine in which direction the extent should be aligned. To the nearest border, inwards or outwards

Value

SpatExtent

See Also

[ext](#), [draw](#)

Examples

```
r <- rast()
e <- ext(-10.1, 9.9, -20.1, 19.9)
ea <- align(e, r)
e
ext(r)
ea

align(e, 0.5)
```

all.equal	<i>Compare two SpatRasters for equality</i>
-----------	---

Description

Compare two SpatRasters for (near) equality.

First the attributes of the objects are compared. If these are the same, a (perhaps small) sample of the raster cells is compared as well.

The sample size used can be increased with the maxcell argument. You can set it to Inf, but for large rasters your computer may not have sufficient memory. See the examples for a safe way to compare all values.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
all.equal(target, current, maxcell=10000, ...)
```

Arguments

target	SpatRaster
current	SpatRaster
maxcell	positive integer. The size of the regular sample used to compare cell values
...	additional arguments passed to all.equal.numeric to compare cell values

Value

Either TRUE or a character vector describing the differences between target and current.

See Also

[compareGeom](#)

Examples

```
x <- sqrt(1:100)
mat <- matrix(x, 10, 10)
r1 <- rast(nrows=10, ncols=10, xmin=0, vals = x)
r2 <- rast(nrows=10, ncols=10, xmin=0, vals = mat)

all.equal(r1, r2)
all.equal(r1, r1*1)
all.equal(rast(r1), rast(r2))

# compare geometries
compareGeom(r1, r2)

# Compare all cell values for near equality
```

```
# as floating point number imprecision can be a problem
m <- minmax(r1 - r2)
all(abs(m) < 1e-7)

# comparison of cell values to create new SpatRaster
e <- r1 == r2
```

animate

*Animate a SpatRaster***Description**

Animate (sequentially plot) the layers of a SpatRaster to create a movie.

This does not work with R-Studio.

Usage

```
## S4 method for signature 'SpatRaster'
animate(x, pause=0.25, main, range, maxcell=50000, n=1, ...)
```

Arguments

x	SpatRaster
pause	numeric. How long should be the pause be between layers?
main	title for each layer. If not supplied the z-value is used if available. Otherwise the names are used.
range	numeric vector of length 2. Range of values to plot
maxcell	integer > 0. Maximum number of cells to use for the plot. If maxcell < ncell(x), spatSample(type="regular") is used before plotting
n	integer > 0. Number of loops
...	Additional arguments passed to plot

Value

None

See Also

[plot](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
animate(s, n=1)
```

Description

Apply a function to the values of each cell of a `SpatRaster`. Similar to [apply](#) – think of each layer in a `SpatRaster` as a column (or row) in a matrix.

This is generally used to summarize the values of multiple layers into one layer; but this is not required.

`app` calls function `fun` with the raster data as first argument. Depending on the function supplied, the raster data is represented as either a matrix in which each layer is a column, or a vector representing a cell. The function should return a vector or matrix that is divisible by `ncell(x)`. Thus, both "sum" and "rowSums" can be used, but "colSums" cannot be used.

You can also apply a function `fun` across datasets by layer of a `SpatRasterDataset`. In that case, summarization is by layer across `SpatRasters`.

Usage

```
## S4 method for signature 'SpatRaster'
app(x, fun, ..., cores=1, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterDataset'
app(x, fun, ..., cores=1, filename="", overwrite=FALSE, wopt=list())
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatRasterDataset</code>
<code>fun</code>	a function that operates on a vector or matrix. This can be a function that is defined in base-R or in a package, or a function you write yourself (see examples). Functions that return complex output (e.g. a list) may need to be wrapped in your own function to simplify the output to a vector or matrix. The following functions have been re-implemented in C++ for speed: "sum", "mean", "median", "modal", "which", "which.min", "which.max", "min", "max", "prod", "any", "all", "sd", "std", "first". To use the base-R function for say, "min", you could use something like <code>fun=function(i) min(i)</code> or the equivalent <code>fun = \(i) min(i)</code>
<code>...</code>	additional arguments for <code>fun</code> . These are typically numerical constants. They should <i>never</i> be another <code>SpatRaster</code>
<code>cores</code>	positive integer. If <code>cores > 1</code> , a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under <code>fun</code>)
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster

Details

To speed things up, parallelization is supported, but this is often not helpful, and it may actually be slower. There is only a speed gain if you have many cores (> 8) and/or a very complex (slow) function `fun`. If you write `fun` yourself, consider supplying a `cppFunction` made with the `Rcpp` package instead (or go have a cup of tea while the computer works for you).

Value

`SpatRaster`

See Also

[lapp](#), [tapp](#), [Math-methods](#), [roll](#)

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
x <- c(r, sqrt(r), r+50)
s <- app(x, fun=sum)
s
# for a few generic functions like
# "sum", "mean", and "max" you can also do
sum(x)

## SpatRasterDataset
sd <- sds(x, x*2, x/3)
a <- app(sd, max)
a
# same as
max(x, x*2, x/3)
# and as (but slower)
b <- app(sd, function(i) max(i))

## also works for a single layer
f <- function(i) (i+1) * 2 * i + sqrt(i)
s <- app(r, f)
# same as above, but that is not memory-safe
# and has no filename argument
s <- f(r)

## Not run:
#### multiple cores
test0 <- app(x, sqrt)
test1 <- app(x, sqrt, cores=2)

testfun <- function(i) { 2 * sqrt(i) }
test2 <- app(x, fun=testfun, cores =2)

## this fails because testfun is not exported to the nodes
# test3 <- app(x, fun=function(i) testfun(i), cores=2)
```

```
## to export it, add it as argument to fun
test3 <- app(x, fun=function(i, ff) ff(i), cores =3, ff=testfun)

## End(Not run)
```

approximate	<i>Estimate values for cell values that are NA by interpolating between layers</i>
-------------	--

Description

approximate uses the stats function [approx](#) to estimate values for cells that are NA by interpolation across layers. Layers are considered equidistant, unless argument z is used, or time(x) returns values that are not NA, in which case these values are used to determine distance between layers.

For estimation based on neighboring cells see [focal](#)

Usage

```
## S4 method for signature 'SpatRaster'
approximate(x, method="linear", yleft, yright,
            rule=1, f=0, ties=mean, z=NULL, NArule=1,filename="", ...)
```

Arguments

x	SpatRaster
method	specifies the interpolation method to be used. Choices are "linear" or "constant" (step function; see the example in approx)
yleft	the value to be returned before a non-NA value is encountered. The default is defined by the value of rule given below
yright	the value to be returned after the last non-NA value is encountered. The default is defined by the value of rule given below
rule	an integer (of length 1 or 2) describing how interpolation is to take place at for the first and last cells (before or after any non-NA values are encountered). If rule is 1 then NAs are returned for such points and if it is 2, the value at the closest data extreme is used. Use, e.g., rule = 2:1, if the left and right side extrapolation should differ
f	for method = "constant" a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If y0 and y1 are the values to the left and right of the point then the value is $y_0 \cdot (1-f) + y_1 \cdot f$ so that $f = 0$) is right-continuous and $f = 1$ is left-continuous
ties	Handling of tied 'z' values. Either a function with a single vector argument returning a single number result or the string "ordered"
z	numeric vector to indicate the distance between layers (e.g., depth). The default is time(x) if these are not NA or else 1:nlys(x)

NArule	single integer used to determine what to do when only a single layer with a non-NA value is encountered (and linear interpolation is not possible). The default value of 1 indicates that all layers will get this value for that cell; all other values do not change the cell values
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also[focal](#), [fillTime](#)**Examples**

```

r <- rast(ncols=5, nrows=5)
r1 <- setValues(r, runif(ncell(r)))
r2 <- setValues(r, runif(ncell(r)))
r3 <- setValues(r, runif(ncell(r)))
r4 <- setValues(r, runif(ncell(r)))
r5 <- setValues(r, NA)
r6 <- setValues(r, runif(ncell(r)))
r1[6:10] <- NA
r2[5:15] <- NA
r3[8:25] <- NA
s <- c(r1,r2,r3,r4,r5,r6)
s[1:5] <- NA
x1 <- approximate(s)
x2 <- approximate(s, rule=2)
x3 <- approximate(s, rule=2, z=c(1,2,3,5,14,15))

```

Arith-methods

*Arithmetic***Description**

Standard arithmetic operators for computations with SpatRasters. Computations are local (applied on a cell by cell basis). If multiple SpatRaster objects are used, these must have the same geometry (extent and resolution). These operators have been implemented:

`+`, `-`, `*`, `/`, `^`, `%%`, `%/%`

You can also use a SpatRaster and a vector or a matrix. If you use a SpatRaster with a vector of multiple numbers, each element in the vector is considered a layer (with a constant value). If you use a SpatRaster with a matrix, the number of columns of the matrix must match the number of layers of the SpatRaster. The rows are used to match the cells. That is, if there are two rows, these match cells 1 and 2, and they are recycled to 3 and 4, etc.

The following methods have been implemented for (SpatExtent, SpatExtent): +, -, and the following for (SpatExtent, numeric): +, -, *, /, %%

Value

SpatRaster or SpatExtent

See Also

[ifel](#) to conveniently combine operations and [Math-methods](#) or [app](#) to use mathematical functions not implemented by the package.

Examples

```
r1 <- rast(ncols=10, nrows=10)
v <- runif(ncell(r1))
v[10:20] <- NA
values(r1) <- v
r2 <- rast(r1)
values(r2) <- 1:ncell(r2) / ncell(r2)
r3 <- r1 + r2
r2 <- r1 / 10
r3 <- r1 * (r2 - 1 / r2)

b <- c(r1, r2, r3)
b2 <- b * 10

### SpatExtent methods
x <- ext(0.1, 2.2, 0, 3)
y <- ext(-2, 1, -2,2)
# union
x + y
# intersection
x * y

e <- x
e
e * 2
e / 2
e + 1
e - 1
```

as.character

Create a text representation of (the skeleton of) an object

Description

Create a text representation of (the skeleton of) an object

Usage

```
## S4 method for signature 'SpatExtent'
as.character(x)

## S4 method for signature 'SpatRaster'
as.character(x)
```

Arguments

x SpatRaster

Value

character

Examples

```
r <- rast()
ext(r)
ext(c(0, 20, 0, 20))
```

as.data.frame

SpatRaster or SpatVector to data.frame

Description

Coerce a SpatRaster or SpatVector to a data.frame

Usage

```
## S4 method for signature 'SpatVector'
as.data.frame(x, row.names=NULL, optional=FALSE, geom=NULL, ...)

## S4 method for signature 'SpatRaster'
as.data.frame(x, row.names=NULL, optional=FALSE, xy=FALSE, cells=FALSE, na.rm=NA, ...)
```

Arguments

x	SpatRaster or SpatVector
geom	character or NULL. If not NULL, either "WKT" or "HEX", to get the geometry included in Well-Known-Text or hexadecimal notation. If x has point geometry, it can also be "XY" to add the coordinates of each point
xy	logical. If TRUE, the coordinates of each raster cell are included
cells	logical. If TRUE, the cell numbers of each raster cell are included
na.rm	logical. If TRUE, cells that have a NA value in at least one layer are removed. If the argument is set to NA only cells that have NA values in all layers are removed

...	Additional arguments passed to the data.frame
row.names	This argument is ignored
optional	This argument is ignored

Value

data.frame

See Also

[as.list](#), [as.matrix](#). See [geom](#) to only extract the geometry of a [SpatVector](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
as.data.frame(v)
```

as.list	<i>Coerce a Spat* object to a list</i>
---------	--

Description

Coerce a [SpatRaster](#), [SpatRasterCollection](#), [SpatRasterDataset](#), [SpatVector](#) or [SpatVectorCollection](#) to a list. With a [SpatRaster](#), each layer becomes a list element. With a [SpatRasterCollection](#) or [SpatRasterDataset](#), each [SpatRaster](#) becomes a list element. With a [SpatVector](#), each variable (attribute) becomes a list element. With a [SpatVectorCollection](#), each [SpatVector](#) becomes a list element.

Usage

```
## S4 method for signature 'SpatRaster'
as.list(x, ...)

## S4 method for signature 'SpatRasterCollection'
as.list(x, ...)

## S4 method for signature 'SpatVector'
as.list(x, geom=NULL, ...)

## S4 method for signature 'SpatVectorCollection'
as.list(x, ...)
```

Arguments

x	SpatRaster , SpatRasterDataset , SpatRasterCollection , or SpatVector
geom	character or NULL. If not NULL, either "WKT" or "HEX", to get the geometry included in Well-Known-Text or hexadecimal notation. If x has point geometry, it can also be "XY" to add the coordinates of each point
...	Additional arguments. These are ignored

Value

list

See Also

see [coerce](#) for as.data.frame with a SpatRaster; and [geom](#) to only extract the geometry of a SpatVector

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
as.list(v)
```

```
s <- rast(system.file("ex/logo.tif", package="terra")) + 1
as.list(s)
```

as.raster

*Coerce to a "raster" object***Description**

Implementation of the generic [as.raster](#) function to create a "raster" (small r) object. Such objects can be used for plotting with the [rasterImage](#) function. NOT TO BE CONFUSED with the Raster* (big R) objects defined by the 'raster' package!

Usage

```
## S4 method for signature 'SpatRaster'
as.raster(x, maxcell=500000, col)
```

Arguments

x	SpatRaster
maxcell	positive integer. Maximum number of cells to use for the plot
col	vector of colors. Default is col=rev(terrain.colors(255)))

Value

'raster' object

Examples

```
r <- rast(ncols=3, nrows=3)
values(r) <- 1:ncell(r)
as.raster(r)
```

as.spatvector	<i>Conversion to a SpatVector, or to another SpatVector type</i>
---------------	--

Description

Conversion of a SpatRaster or SpatExtent to a SpatVector of points, lines, or polygons;
And conversion of a SpatVector to another SpatVector type.

Usage

```
## S4 method for signature 'SpatRaster'
as.polygons(x, trunc=TRUE, dissolve=TRUE, values=TRUE,
na.rm=TRUE, na.all=FALSE, extent=FALSE)

## S4 method for signature 'SpatRaster'
as.lines(x)

## S4 method for signature 'SpatRaster'
as.points(x, values=TRUE, na.rm=TRUE, na.all=FALSE)

## S4 method for signature 'SpatVector'
as.polygons(x, extent=FALSE)

## S4 method for signature 'SpatVector'
as.lines(x)

## S4 method for signature 'SpatVector'
as.points(x, multi=FALSE, skiplast=TRUE)

## S4 method for signature 'SpatExtent'
as.polygons(x, crs="")

## S4 method for signature 'SpatExtent'
as.lines(x, crs="")

## S4 method for signature 'SpatExtent'
as.points(x, crs="")
```

Arguments

x	SpatRaster or SpatVector
trunc	logical; truncate values to integers. Cells with the same value are merged. Therefore, if trunc=FALSE the object returned can have many cells and can be very large
dissolve	logical; combine cells with the same values? If TRUE only the first layer in x is processed

values	logical; include cell values as attributes?
multi	logical. If TRUE a multi-point geometry is returned
skiplast	logical. If TRUE the last point of a polygon (which is the same as the first point) is not included
extent	logical. if TRUE, a polygon for the extent of the SpatRaster or SpatVector is returned. If x is a SpatRaster, the polygon has vertices for each row and column, not just the four corners of the raster. This can be useful for more precise projection. If that is not required, it is more efficient to get the extent represented by only the four corners with <code>as.polygons(ext(x), crs=crs(x))</code>
na.rm	logical. If TRUE cells that are NA are ignored
na.all	logical. If TRUE cells are only ignored if na.rm=TRUE and their value is NA for all layers instead of for any layer
crs	character. The coordinate reference system (see crs)

Value

SpatVector

Examples

```
r <- rast(ncols=2, nrows=2)
values(r) <- 1:ncell(r)

as.points(r)
as.lines(ext(r), crs=crs(r))

if (gdal() >= "3.0.0") {
  p <- as.polygons(r)
  p
  as.lines(p)
  as.points(p)
}
```

atan2

Two argument arc-tangent

Description

For SpatRasters x and y, `atan2(y, x)` returns the angle in radians for the tangent y/x, handling the case when x is zero. See [Trig](#)

See [Math-methods](#) for other trigonometric and mathematical functions that can be used with SpatRasters.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
atan2(y, x)

## S4 method for signature 'SpatRaster,SpatRaster'
atan_2(y, x, filename, ...)
```

Arguments

y	SpatRaster
x	SpatRaster
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

See Also

[Math-methods](#)

Examples

```
r1 <- rast(nrows=10, ncols=10)
r2 <- rast(nrows=10, ncols=10)
values(r1) <- (runif(ncell(r1))-0.5) * 10
values(r2) <- (runif(ncell(r1))-0.5) * 10
atan2(r1, r2)
```

autocorrelation	<i>Spatial autocorrelation</i>
-----------------	--------------------------------

Description

Compute spatial autocorrelation for a numeric vector or a SpatRaster. You can compute standard (global) Moran's I or Geary's C, or local indicators of spatial autocorrelation (Anselin, 1995).

Usage

```
## S4 method for signature 'numeric'
autocor(x, w, method="moran")

## S4 method for signature 'SpatRaster'
autocor(x, w=matrix(c(1,1,1,1,0,1,1,1,1),3), method="moran", global=TRUE)
```

Arguments

x	numeric or SpatRaster
w	Spatial weights defined by or a rectangular matrix. For a SpatRaster this matrix must the sides must have an odd length (3, 5, ...)
global	logical. If TRUE global autocorrelation is computed instead of local autocorrelation
method	character. If x is numeric or SpatRaster: "moran" for Moran's I and "geary" for Geary's C. If x is numeric also: "Gi", "Gi*" (the Getis-Ord statistics), locmor (local Moran's I) and "mean" (local mean)

Details

The default setting uses a 3x3 neighborhood to compute "Queen's case" indices. You can use a filter (weights matrix) to do other things, such as "Rook's case", or different lags.

Value

numeric or SpatRaster

References

- Moran, P.A.P., 1950. Notes on continuous stochastic phenomena. *Biometrika* 37:17-23
- Geary, R.C., 1954. The contiguity ratio and statistical mapping. *The Incorporated Statistician* 5: 115-145
- Anselin, L., 1995. Local indicators of spatial association-LISA. *Geographical Analysis* 27:93-115
- https://en.wikipedia.org/wiki/Indicators_of_spatial_association

See Also

The spdep package for additional and more general approaches for computing spatial autocorrelation

Examples

```
### raster
r <- rast(nrows=10, ncols=10, xmin=0)
values(r) <- 1:ncell(r)

autocor(r)

# rook's case neighbors
f <- matrix(c(0,1,0,1,0,1,0,1,0), nrow=3)
autocor(r, f)

# local
rc <- autocor(r, w=f, global=FALSE)

### numeric (for vector data)
```



```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- relate(v, relation="touches")

# global
autocor(v$AREA, w)

# local
v$Gi <- autocor(v$AREA, w, "Gi")
plot(v, "Gi")
```

barplot

Bar plot of a SpatRaster

Description

Create a barplot of the values of a the first layer of a SpatRaster. For large datasets a regular sample with a size of approximately maxcells is used.

Usage

```
## S4 method for signature 'SpatRaster'
barplot(height, maxcell=1000000, digits=0, breaks=NULL, col, ...)
```

Arguments

height	SpatRaster
maxcell	integer. To regularly subsample very large datasets
digits	integer used to determine how to round the values before tabulating. Set to NULL or to a large number if you do not want any rounding
breaks	breaks used to group the data as in cut
col	a color generating function such as rainbow (the default), or a vector of colors
...	additional arguments for plotting as in barplot

Value

A numeric vector (or matrix, when beside = TRUE) of the coordinates of the bar midpoints, useful for adding to the graph. See [barplot](#)

See Also

[hist](#), [boxplot](#)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
barplot(r, digits=-1, las=2, ylab="Frequency")

op <- par(no.readonly = TRUE)
par(mai = c(1, 2, .5, .5))
barplot(r, breaks=10, col=c("red", "blue"), horiz=TRUE, digits=NULL, las=1)
par(op)
```

boundaries	<i>Detect boundaries (edges)</i>
------------	----------------------------------

Description

Detect boundaries (edges). Boundaries are cells that have more than one class in the 4 or 8 cells surrounding it, or, if `classes=FALSE`, cells with values and cells with NA.

Usage

```
## S4 method for signature 'SpatRaster'
boundaries(x, classes=FALSE, inner=TRUE,
           directions=8, falseval=0, filename="", ...)
```

Arguments

<code>x</code>	SpatRaster
<code>inner</code>	logical. If TRUE, "inner" boundaries are returned, else "outer" boundaries are returned
<code>classes</code>	character. Logical. If TRUE all different values are (after rounding) distinguished, as well as NA. If FALSE (the default) only edges between NA and non-NA cells are considered
<code>directions</code>	integer. Which cells are considered adjacent? Should be 8 (Queen's case) or 4 (Rook's case)
<code>falseval</code>	numeric. The value to use for cells that are not a boundary and not NA
<code>filename</code>	character. Output filename
<code>...</code>	options for writing files as in writeRaster

Value

SpatRaster. Cell values are either 1 (a border) or 0 (not a border), or NA

See Also

[focal](#), [patches](#)

Examples

```

r <- rast(nrows=18, ncols=36, xmin=0)
r[150:250] <- 1
r[251:450] <- 2
bi <- boundaries(r)
bo <- boundaries(r, inner=FALSE)
bc <- boundaries(r, classes=TRUE)
#plot(bc)

```

boxplot

*Box plot of SpatRaster data***Description**

Box plot of layers in a SpatRaster

Usage

```

## S4 method for signature 'SpatRaster'
boxplot(x, y=NULL, maxcell=100000, ...)

```

Arguments

<code>x</code>	SpatRaster
<code>y</code>	NULL or a SpatRaster. If <code>x</code> is a SpatRaster it used to group the values of <code>x</code> by "zone"
<code>maxcell</code>	Integer. Number of cells to sample from datasets
<code>...</code>	additional arguments passed to <code>graphics::boxplot</code>

Value

boxplot returns a list (invisibly) that can be used with [bxp](#)

See Also

[pairs](#), [hist](#)

Examples

```

r1 <- r2 <- r3 <- rast(ncols=10, nrows=10)
set.seed(409)
values(r1) <- rnorm(ncell(r1), 100, 40)
values(r2) <- rnorm(ncell(r1), 80, 10)
values(r3) <- rnorm(ncell(r1), 120, 30)
s <- c(r1, r2, r3)
names(s) <- c("Apple", "Pear", "Cherry")

boxplot(s, notch=TRUE, col=c("red", "blue", "orange"), main="Box plot", ylab="random", las=1)

```

```

op <- par(no.readonly = TRUE)
par(mar=c(4,6,2,2))
boxplot(s, horizontal=TRUE, col="lightskyblue", axes=FALSE)
axis(1)
axis(2, at=0:3, labels=c("", names(s)), las=1, cex.axis=.9, lty=0)
par(op)

## boxplot with 2 layers
v <- vect(system.file("ex/lux.shp", package="terra"))
r <- rast(system.file("ex/elev.tif", package="terra"))
y <- rasterize(v, r, "NAME_2")
b <- boxplot(r, y)
bxp(b)

```

buffer

*Create a buffer around vector geometries or raster patches***Description**

Calculate a buffer around all cells that are not NA in a `SpatRaster`, or around the geometries of a `SpatVector`

`SpatRaster` cells inside the buffer distance get a value of 1.

Note that the distance unit of the buffer width parameter is meters if the CRS is (+proj=longlat), and in map units (typically also meters) if not.

Usage

```

## S4 method for signature 'SpatRaster'
buffer(x, width, background=0, filename="", ...)

## S4 method for signature 'SpatVector'
buffer(x, width, quadsegs=10, capstyle="round",
joinstyle="round", mitrelimit=NA, singlesided=FALSE)

```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>width</code>	numeric. Unit is meter if <code>x</code> has a longitude/latitude CRS, or in the units of the coordinate reference system in other cases (typically also meter). The value should be > 0 if <code>x</code> is a <code>SpatRaster</code> . If <code>x</code> is a <code>SpatVector</code> , this argument is vectorized, meaning that you can provide a different value for each geometry in <code>x</code> ; and you can also use the name of a variable in <code>x</code> that has the widths
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster
<code>background</code>	numeric. value to assign to cells outside the buffer. If this value is zero or FALSE, a boolean <code>SpatRaster</code> is returned

quadsegs	positive integer. Number of line segments to use to draw a quart circle
capstyle	character. One of "round", "square" or "flat". Ignored if <code>is.lonlat(x)</code>
joinstyle	character. One of "round", "mitre" or "bevel". Ignored if <code>is.lonlat(x)</code>
mitrelimit	numeric. Place an upper bound on a mitre join to avoid it from extending very far from acute angles in the input geometry. Ignored if <code>is.lonlat(x)</code>
singlesided	logical. If TRUE a buffer is constructed on only one side of each input line. Ignored if <code>is.lonlat(x)</code>

Value

Same as `x`

See Also

[distance](#), [elongate](#)

Examples

```
r <- rast(ncols=36, nrows=18)
r[500] <- 1
b <- buffer(r, width=5000000)
plot(b)

v <- vect(rbind(c(10,10), c(0,60)), crs="+proj=merc")
b <- buffer(v, 20)
plot(b)
points(v)

crs(v) <- "+proj=longlat"
b <- buffer(v, 1500000)
plot(b)
points(v)
```

Description

With `c` you can:

- Combine *SpatRaster* objects. They must have the same extent and resolution. However, if `x` is empty (has no cell values), its geometry is ignored with a warning. Two empty *SpatRasters* with the same geometry can also be combined (to get a summed number of layers). Also see [add<-](#)
 - Add a *SpatRaster* to a *SpatRasterDataset*
 - Add *SpatVector* objects to a new or existing *SpatVectorCollection*
- To append *SpatVectors*, use `rbind`.

Usage

```
## S4 method for signature 'SpatRaster'
c(x, ..., warn=TRUE)

## S4 method for signature 'SpatRasterDataset'
c(x, ...)

## S4 method for signature 'SpatVector'
c(x, ...)

## S4 method for signature 'SpatVectorCollection'
c(x, ...)
```

Arguments

x	SpatRaster, SpatVector, SpatRasterDataset or SpatVectorCollection
warn	logical. If TRUE, a warning is emitted if x is an empty SpatRaster
...	as for x (you can only combine raster with raster data and vector with vector data)

Value

Same class as x

See Also

[add<-](#)

Examples

```
r <- rast(nrows=5, ncols=9)
values(r) <- 1:ncell(r)
x <- c(r, r*2, r*3)
```

cartogram

Cartogram

Description

Make a cartogram, that is, a map where the area of polygons is made proportional to another variable. This can be a good way to map raw count data (e.g. votes).

Usage

```
## S4 method for signature 'SpatVector'
cartogram(x, var, type)
```

Arguments

x	SpatVector
var	character. A variable name in x
type	character. Cartogram type, only "nc" (non-contiguous) is currently supported

Value

SpatVector

See Also

[plot](#), [rescale](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$value <- 1:12
p <- cartogram(v, "value", "nc")
plot(v, col="light gray", border="gray")
lines(p, col="red", lwd=2)
```

catalyze

Factors to numeric

Description

Change a categorical layer into one or more numerical layers. With `as.numeric` you can transfer the active category values to cell values in a non-categorical `SpatRaster`. `catalyze` creates new layers for each category.

Usage

```
## S4 method for signature 'SpatRaster'
as.numeric(x, index=NULL, filename="", ...)
```

```
## S4 method for signature 'SpatRaster'
catalyze(x, filename="", ...)
```

Arguments

x	SpatRaster
index	positive integer, indicating the column in data.frame value to be used as the category, skipping the first column with the ID. If NULL the active category is used
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[activeCat](#), [cats](#)

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE) + 10
d <- data.frame(id=11:13, cover=c("forest", "water", "urban"), letters=letters[1:3], value=10:12)
levels(r) <- d
catalyze(r)

activeCat(r) <- 3
as.numeric(r)
```

cells	<i>Get cell numbers</i>
-------	-------------------------

Description

Get the cell numbers covered by a SpatVector or SpatExtent. Or that match values in a vector; or all non NA values.

Usage

```
## S4 method for signature 'SpatRaster,missing'
cells(x, y)

## S4 method for signature 'SpatRaster,numeric'
cells(x, y)

## S4 method for signature 'SpatRaster,SpatVector'
cells(x, y, method="simple", weights=FALSE, exact=FALSE, touches=is.lines(y))

## S4 method for signature 'SpatRaster,SpatExtent'
cells(x, y)
```

Arguments

- x SpatRaster
- y SpatVector, SpatExtent, 2-column matrix representing points, numeric representing values to match, or missing

method	character. Method for getting cell numbers for points. The default is "simple", the alternative is "bilinear". If it is "bilinear", the four nearest cells and their weights are returned
weights	logical. If TRUE and y has polygons, the approximate fraction of each cell that is covered is returned as well
exact	logical. If TRUE and y has polygons, the exact fraction of each cell that is covered is returned as well
touches	logical. If TRUE, values for all cells touched by lines or polygons are extracted, not just those on the line render path, or whose center point is within the polygon. Not relevant for points

Value

numeric vector or matrix

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
r[c(1:25, 31:100)] <- NA
r <- ifel(r > 28, r + 10, r)

# all cell numbers of cells that are not NA
cells(r)

# cell numbers that match values
x <- cells(r, c(28,38))
x$lyr.1

# cells for points
m <- cbind(x=c(0,10,-30), y=c(40,-10,20))
cellFromXY(r, m)

v <- vect(m)
cells(r, v)
cells(r, v, method="bilinear")

# cells for polygons
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
r <- rast(v)
cv <- cells(r, v)
```

Description

Compute the area covered by individual raster cells.

Computing the surface area of raster cells is especially relevant for longitude/latitude rasters.

But note that for both angular (longitude/latitude) and for planar (projected) coordinate reference systems raster cells sizes are generally not constant, unless you are using an equal-area coordinate reference system.

For planar CRSs, the area is therefore not computed based on the linear units of the coordinate reference system, but on the **actual** area by transforming cells to longitude/latitude. If you do not want that correction, you can use `transform=FALSE` or `init(x, prod(res(x)))`

Usage

```
## S4 method for signature 'SpatRaster'
cellSize(x, mask=FALSE, lyrs=FALSE, unit="m", transform=TRUE, rcx=100, filename="", ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>mask</code>	logical. If TRUE, cells that are NA in <code>x</code> are also NA in the output
<code>lyrs</code>	logical. If TRUE and <code>mask=TRUE</code> , the output has the same number of layers as <code>x</code> . That is only useful if cases where the layers of <code>x</code> have different cells that are NA
<code>unit</code>	character. One of "m", "km", or "ha"
<code>transform</code>	logical. If TRUE, planar CRS data are transformed to lon/lat for accuracy
<code>rcx</code>	positive integer. The maximum number of rows and columns to be used to compute area of planar data if <code>transform=TRUE</code> . If <code>x</code> has more rows and/or columns, the raster is aggregated to match this limit, and values for the original cells are estimated by bilinear interpolation (see <code>resample</code>). This can save a lot of time
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster

Value

`SpatRaster`

See Also

[expand](#)

Examples

```
# SpatRaster
r <- rast(nrows=18, ncols=36)
v <- 1:ncell(r)
v[200:400] <- NA
```

```

values(r) <- v

# size of each raster cell
a <- cellSize(r)

# illustration of distortion
r <- rast(ncols=90, nrows=45, ymin=-80, ymax=80)
m <- project(r, "+proj=merc")

bad <- init(m, prod(res(m)) / 1000000, names="naive")
good <- cellSize(m, unit="km", names="corrected")
plot(c(good, bad), nc=1, mar=c(2,2,1,6))

```

centroids

*Centroids***Description**

Get the centroids of polygons or lines, or centroid-like points that are guaranteed to be inside the polygons or on the lines.

Usage

```

## S4 method for signature 'SpatVector'
centroids(x, inside=FALSE)

```

Arguments

x	SpatVector
inside	logical. If TRUE the points returned are guaranteed to be inside the polygons or on the lines, but they are not the true centroids. True centroids may be outside a polygon, for example when a polygon is "bean shaped", and they are unlikely to be on their line

Value

SpatVector of points

Examples

```

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
x <- centroids(v)
y <- centroids(v, TRUE)

```

 clamp

Clamp values

Description

Clamp values to a minimum and maximum value. That is, all values below a lower threshold value and above the upper threshold value become either NA, or, if values=TRUE, become the threshold value

Usage

```
## S4 method for signature 'SpatRaster'
clamp(x, lower=-Inf, upper=Inf, values=TRUE, filename="", ...)

## S4 method for signature 'numeric'
clamp(x, lower=-Inf, upper=Inf, values=TRUE, ...)
```

Arguments

x	SpatRaster
lower	numeric with the lowest acceptable value (you can specify a different value for each layer). Or a SpatRaster that has a single layer or the same number of layers as x
upper	numeric with the highest acceptable value (you can specify a different value for each layer). Or a SpatRaster that has a single layer or the same number of layers as x
values	logical. If FALSE values outside the clamping range become NA, if TRUE, they get the extreme values
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[classify](#)

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
rc <- clamp(r, 25, 75)
rc
```

clamp_ts	<i>clamp time series data</i>
----------	-------------------------------

Description

clamp time-series data that are S shaped. The value in layers before the minimum value in a cell can be set to that minimum value, and the value in layers after the maximum value for a cell can be set to that maximum value.

Usage

```
## S4 method for signature 'SpatRaster'
clamp_ts(x, min=FALSE, max=TRUE, filename="", ...)
```

Arguments

x	SpatRaster
min	logical. If TRUE the time-series is clamped to the minimum value
max	logical. If TRUE the time-series is clamped to the maximum value
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[clamp](#), [cummin](#), [cummax](#)

Examples

```
sigm <- function(x) { .8 / (1 + exp(-(x-10))) + runif(length(x))/4 }
r <- rast(ncols=10, nrows=10, nlyr=50)
s <- seq(5.2, 15,.2)
set.seed(1)
values(r) <- t(replicate(100, sigm(s)))

x <- clamp_ts(r, TRUE, TRUE)

plot(unlist(r[4]))
lines(unlist(x[4]))
```

classify	<i>Classify (or reclassify) cell values</i>
----------	---

Description

Classify values of a SpatRaster. The function (re-)classifies groups of values to other values.

The classification is done based on the argument `rc1`. You can classify ranges by specifying a three-column matrix "from-to-becomes" or change specific values by using a two-column matrix "is-becomes". You can also supply a vector with "cuts" or the "number of cuts".

With "from-to-becomes" or "is-becomes" classification is done in the row order of the matrix. Thus, if there are overlapping ranges or values, the first time a number is within a range determines the reclassification value.

With "cuts" the values are sorted, so that the order in which they are provided does not matter.

Usage

```
## S4 method for signature 'SpatRaster'
classify(x, rc1, include.lowest=FALSE, right=TRUE,
         others=NULL, brackets=TRUE, filename="", ...)
```

Arguments

<code>x</code>	SpatRaster
<code>rc1</code>	<p>matrix for classification. This matrix must have 1, 2 or 3 columns. If there are three columns, the first two columns are "from" "to" of the input values, and the third column "becomes" has the new value for that range.</p> <p>The two column matrix ("is", "becomes") can be useful for classifying integer values. In that case, the arguments <code>right</code> and <code>include.lowest</code> are ignored.</p> <p>A single column matrix (or a vector) is interpreted as a set of cuts if there is more than one value. In that case the values are classified based on their location in-between the cut-values.</p> <p>If a single number is provided, that is used to make that number of cuts, at equal intervals between the lowest and highest values of the SpatRaster.</p>
<code>include.lowest</code>	logical, indicating if a value equal to the lowest value in <code>rc1</code> (or highest value in the second column, for <code>right=FALSE</code>) should be included.
<code>right</code>	<p>logical. If <code>TRUE</code>, the intervals are closed on the right (and open on the left). If <code>FALSE</code> they are open at the right and closed at the left. "open" means that the extreme value is <i>*not*</i> included in the interval. Thus, right-closed and left open is $(0, 1] = \{x \mid 0 < x \leq 1\}$. You can also close both sides with <code>right=NA</code>, that is only meaningful if you "from-to-becomes" classification with integers. For example to classify 1-5 -> 1, 6-10 -> 2, 11-15 -> 3. That may be easier to read and write than the equivalent 1-5 -> 1, 5-10 -> 2, 10-15 -> 3 with <code>right=TRUE</code> and <code>include.lowest=TRUE</code></p>
<code>others</code>	numeric. If not <code>NULL</code> all values that are not matched are set to this value. Otherwise they retain their original value.

brackets	logical. If TRUE, intervals are have parenthesis or brackets around them to indicate whether they are open or closed. Only applies if <code>rcl</code> is a vector (or single column matrix)
filename	character. Output filename
...	Additional arguments for writing files as in writeRaster

Value

SpatRaster

Note

`classify` works with the "raw" values of categorical rasters, ignoring the levels (labels, categories). To change the labels of categorical rasters, use [subst](#) instead.

For model-based classification see [predict](#)

See Also

[subst](#) for simpler from-to replacement

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- (0:99)/99

## from-to-becomes
# classify the values into three groups
# all values >= 0 and <= 0.25 become 1, etc.
m <- c(0, 0.25, 1,
      0.25, 0.5, 2,
      0.5, 1, 3)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rcl <- classify(r, rclmat, include.lowest=TRUE)

## cuts
# equivalent to the above, but now a categorical SpatRaster is returned
rc2 <- classify(r, c(0, 0.25, 0.5, 1), include.lowest=TRUE, brackets=TRUE)
freq(rc2)

## is-becomes
x <- round(r*3)
unique(x)
# replace 0 with NA
y <- classify(x, cbind(0, NA))
unique(y)

# multiple replacements
m <- rbind(c(2, 200), c(3, 300))
m

rcx1 <- classify(x, m)
```

```
unique(rcx1)

rcx2 <- classify(x, m, others=NA)
unique(rcx2)
```

click

Query by clicking on a map

Description

Click on a map (plot) to get the coordinates or the values of a `SpatRaster` or `SpatVector` at that location. For a `SpatRaster` you can also get the coordinates and cell number of the location.

This does to work well on the default RStudio plotting device. To work around that, you can first run `dev.new(noRStudioGD = TRUE)` which will create a separate window for plotting, then use `plot()` followed by `click()` and click on the map .

Usage

```
## S4 method for signature 'SpatRaster'
click(x, n=10, id=FALSE, xy=FALSE, cell=FALSE, type="p", show=TRUE, ...)

## S4 method for signature 'SpatVector'
click(x, n=10, id=FALSE, xy=FALSE, type="p", show=TRUE, ...)

## S4 method for signature 'missing'
click(x, n=10, id=FALSE, type="p", show=TRUE, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code> , or missing
<code>n</code>	number of clicks on the plot (map)
<code>id</code>	logical. If TRUE, a numeric ID is shown on the map that corresponds to the row number of the output
<code>xy</code>	logical. If TRUE, xy coordinates are included in the output
<code>cell</code>	logical. If TRUE, cell numbers are included in the output
<code>type</code>	one of "n", "p", "l" or "o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines. See <code>?locator</code>
<code>show</code>	logical. Print the values after each click?
<code>...</code>	additional graphics parameters used if <code>type != "n"</code> for plotting the locations. See <code>?locator</code>

Value

The value(s) of `x` at the point(s) clicked on (or touched by the box drawn). A `data.frame` with the value(s) of all layers of `SpatRaster` `x` for the cell(s) clicked on; or with the attributes of the geometries of `SpatVector` `x` that intersect with the box drawn).

Note

The plot only provides the coordinates for a spatial query, the values are read from the `SpatRaster` or `SpatVector` that is passed as an argument. Thus you can extract values from an object that has not been plotted, as long as it spatially overlaps with the extent of the plot.

Unless the process is terminated prematurely values at at most `n` positions are determined. The identification process can be terminated, depending on how you interact with R, by hitting `Esc`, or by clicking the right mouse button and selecting "Stop" from the menu, or from the "Stop" menu on the graphics window.

See Also

[draw](#)

Examples

```
## Not run:
r <-rast(system.file("ex/elev.tif", package="terra"))
plot(r)
click(r, n=1)
## now click on the plot (map)

## End(Not run)
```

coerce

Coercion of a SpatRaster to a vector, matrix or array

Description

Coercion to other object types

Usage

```
## S4 method for signature 'SpatRaster'
as.vector(x, mode='any')

## S4 method for signature 'SpatRaster'
as.matrix(x, wide=FALSE, ...)

## S4 method for signature 'SpatRaster'
as.array(x)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>wide</code>	logical. If <code>FALSE</code> each layer in the <code>SpatRaster</code> becomes a column in the matrix and each cell in the <code>SpatRaster</code> becomes a row. If <code>TRUE</code> each row in the <code>SpatRaster</code> becomes a row in the matrix and each column in the <code>SpatRaster</code> becomes a column in the matrix

mode this argument is ignored
... additional arguments (none implemented)

Value

vector, matrix, or array

See Also

[as.data.frame](#) and [as.polygons](#)

Examples

```
r <- rast(ncols=2, nrows=2)
values(r) <- 1:ncell(r)

as.vector(r)
as.matrix(r)
as.matrix(r, wide=TRUE)
as.data.frame(r, xy=TRUE)
as.array(r)
```

colors	<i>Color table</i>
--------	--------------------

Description

Get or set color table(s) associated with a SpatRaster. Color tables are used for associating colors with values, for use in mapping (plot).

Usage

```
## S4 method for signature 'SpatRaster'
coltab(x)

## S4 replacement method for signature 'SpatRaster'
coltab(x, layer=1)<-value

## S4 method for signature 'SpatRaster'
has.colors(x)
```

Arguments

x SpatRaster
layer positive integer, the layer number or name
value a two-column data.frame (first column the cell value, the second column the color); a vector of colors (the first one is the color for value 0 and so on); or a three (red,green,blue) or four (alpha) column data.frame also from 0 to n; or NULL to remove the color table

Value

data.frame

Examples

```
r <- rast(ncols=3, nrows=2, vals=1:6)
coltb <- data.frame(value=1:6, col=rainbow(6, end=.9))
coltb

plot(r)

has.colors(r)
coltab(r) <- coltb
plot(r)
has.colors(r)

tb <- coltab(r)
class(tb)
dim(tb[[1]])
```

combineGeoms

Combine geometries

Description

Combine the geometries of one SpatVector with those of another. Geometries can be combined based on overlap, shared boundaries and distance (in that order of operation).

The typical use-case of this method is when you are editing geometries and you have a number of small polygons in one SpatVector that should be part of the geometries of the another SpatVector; perhaps because they were small holes inbetween the borders of two SpatVectors.

To append SpatVectors use 'rbind' and see methods like intersect and union for "normal" polygons combinations.

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
combineGeoms(x, y, overlap=TRUE, boundary=TRUE, distance=TRUE,
append=TRUE, minover=0.1, maxdist=Inf, dissolve=TRUE, erase=TRUE)
```

Arguments

x	SpatVector of polygons
y	SpatVector of polygons geometries that are to be combined with x
overlap	logical. If TRUE, a geometry is combined with the geometry it has most overlap with, if the overlap is above minover
boundary	logical. If TRUE, a geometry is combined with the geometry it has most shared border with

distance	logical. If TRUE, a geometry is combined with the geometry it is nearest to
append	logical. Should remaining geometries be appended to the output? Not relevant if distance=TRUE
minover	numericThe fraction of the geometry in codey that overlaps with a geometry in x. Below this threshold, geometries are not considered overlapping
maxdist	numeric. Geometries further away from each other than this distance (in meters) will not be combined
dissolve	logical. Should internal boundaries be dissolved?
erase	logical. If TRUE no new overlapping areas are created

Value

SpatVector

See Also

[union](#), [erase](#), [intersect](#)
[sharedPaths](#), [erase](#), [intersect](#)

Examples

```
x1 <- vect("POLYGON ((0 0, 8 0, 8 9, 0 9, 0 0))")
x2 <- vect("POLYGON ((10 4, 12 4, 12 7, 11 7, 11 6, 10 6, 10 4))")

y1 <- vect("POLYGON ((5 6, 15 6, 15 15, 5 15, 5 6))")
y2 <- vect("POLYGON ((8 2, 9 2, 9 3, 8 3, 8 2))")
y3 <- vect("POLYGON ((2 6, 3 6, 3 8, 2 8, 2 6))")
y4 <- vect("POLYGON ((2 12, 3 12, 3 13, 2 13, 2 12))")

x <- rbind(x1, x2)
values(x) <- data.frame(xid=1:2)
crs(x) <- "+proj=utm +zone=1"

y <- rbind(y1, y2, y3, y4)
values(y) <- data.frame(yid=letters[1:4])
crs(y) <- "+proj=utm +zone=1"

plot(rbind(x, y), border=c(rep("red",2), rep("blue", 4)), lwd=2)
text(x, "xid")
text(y, "yid")

v <- combineGeoms(x, y)
plot(v, col=c("red", "blue"))

v <- combineGeoms(x, y, boundary=FALSE, maxdist=1, minover=.05)
plot(v, col=rainbow(4))
```

Description

Standard comparison and logical operators for computations with SpatRasters. Computations are local (applied on a cell by cell basis). If multiple SpatRaster objects are used, these must have the same geometry (extent and resolution). These operators have been implemented:

Logical: `!`, `&`, `|`, `isTRUE`, `isFALSE`

Compare: `==`, `!=`, `>`, `<`, `<=`, `>=`, `is.na`, `is.nan`, `is.finite`, `is.infinite`

See [not.na](#) for the inverse of `is.na`, and [noNA](#) to detect cells with missing value across layers.

The compare and logic methods implement these operators in a method that can return NA instead of FALSE and allows for setting an output filename.

The terra package does not distinguish between NA (not available) and NaN (not a number). In most cases this state is represented by NaN.

If you use a SpatRaster with a vector of multiple numbers, each element in the vector is considered a layer (with a constant value). If you use a SpatRaster with a matrix, the number of columns of the matrix must match the number of layers of the SpatRaster. The rows are used to match the cells. That is, if there are two rows, these match cells 1 and 2, and they are recycled to 3 and 4, etc.

The following method has been implemented for (**SpatExtent**, **SpatExtent**): `==`

Usage

```
## S4 method for signature 'SpatRaster'
compare(x, y, oper, falseNA=FALSE, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatRaster'
logic(x, oper, falseNA=FALSE, filename="", overwrite=FALSE, ...)
```

Arguments

<code>x</code>	SpatRaster
<code>y</code>	SpatRaster or numeric
<code>oper</code>	character. Operator name. For compare this can be one of <code>"=="</code> , <code>"!="</code> , <code>">"</code> , <code>"<"</code> , <code>">="</code> , <code>"<="</code> and for logic it can be one of <code>"!"</code> , <code>"is.na"</code> , <code>"allNA"</code> , <code>"noNA"</code> , <code>"is.infinite"</code> , <code>"is.finite"</code> , <code>"isTRUE"</code> , <code>"isFALSE"</code>
<code>falseNA</code>	logical. Should the result be TRUE, NA instead of TRUE, FALSE?
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>...</code>	additional arguments for writing files as in writeRaster

Value

SpatRaster or SpatExtent

See Also

[all.equal](#), [Arith-methods](#). See [ifel](#) to conveniently combine operations and [Math-methods](#) or [app](#) to apply any R function to a SpatRaster.

Examples

```
r1 <- rast(ncols=10, nrows=10)
values(r1) <- runif(ncell(r1))
r1[10:20] <- NA
r2 <- rast(r1)
values(r2) <- 1:ncell(r2) / ncell(r2)

x <- is.na(r1)
!x
r1 == r2
compare(r1, r2, "==")
compare(r1, r2, "==", TRUE)
```

compareGeom

Compare geometries of SpatRasters

Description

Evaluate whether two SpatRasters have the same extent, number of rows and columns, projection, resolution, and origin (or a subset of these comparisons). Or whether two SpatVectors have the same geometries, or whether a SpatVector has duplicated geometries.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
compareGeom(x, y, ..., lyrs=FALSE, crs=TRUE, warncrs=FALSE, ext=TRUE,
rowcol=TRUE, res=FALSE, stopOnError=TRUE, messages=FALSE)
```

```
## S4 method for signature 'SpatVector,SpatVector'
compareGeom(x, y, tolerance=0)
```

```
## S4 method for signature 'SpatVector,missing'
compareGeom(x, y, tolerance=0)
```

Arguments

x	SpatRaster
y	SpatRaster
...	Additional SpatRasters
lyrs	logical. If TRUE, the number of layers is compared
crs	logical. If TRUE, coordinate reference systems are compared

warnrcs	logical. If TRUE, a warning is given if the crs is different (instead of an error)
ext	logical. If TRUE, bounding boxes are compared
rowcol	logical. If TRUE, number of rows and columns of the objects are compared
res	logical. If TRUE, resolutions are compared (redundant when checking extent and rowcol)
stopOnError	logical. If TRUE, code execution stops if raster do not match
messages	logical. If TRUE, warning/error messages are printed even if stopOnError=FALSE
tolerance	numeric

Value

logical (SpatRaster) or matrix of logical (SpatVector)

Examples

```
r1 <- rast()
r2 <- rast()
r3 <- rast()
compareGeom(r1, r2, r3)
nrow(r3) <- 10
```

```
## Not run:
compareGeom(r1, r3)
```

```
## End(Not run)
```

concat	<i>Concatenate categorical rasters</i>
--------	--

Description

Combine two categorical rasters by concatenating their levels.

Usage

```
## S4 method for signature 'SpatRaster'
concat(x, y, filename="", ...)
```

Arguments

x	SpatRaster (with a single, categorical, layer)
y	SpatRaster (with a single, categorical, layer)
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[cats](#)

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE)
levels(r) <- data.frame(id=1:3, cover=c("forest", "water", "urban"))

rr <- rast(r)
values(rr) <- sample(1:3, ncell(rr), replace=TRUE)
levels(rr) <- data.frame(id=c(1:3), color=c("red", "green", "blue"))

x <- concats(r, rr)
x
levels(x)[[1]]
```

contour	<i>Contour plot</i>
---------	---------------------

Description

Contour lines of a SpatRaster. Use add=TRUE to add the lines to the current plot. See [contour](#) for details.

if filled=TRUE, a new filled contour plot is made. See [filled.contour](#) for details.

as.contour returns the contour lines as a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
contour(x, maxcells=100000, filled=FALSE, ...)

## S4 method for signature 'SpatRaster'
as.contour(x, maxcells=100000, ...)
```

Arguments

x	SpatRaster. Only the first layer is used
maxcells	maximum number of pixels used to create the contours
filled	logical. If TRUE, a filled.contour plot is made
...	any argument that can be passed to contour or filled.contour (graphics package)

See Also[plot](#)**Examples**

```

r <- rast(system.file("ex/elev.tif", package="terra"))
plot(r)
contour(r, add=TRUE)

v <- as.contour(r)
plot(r)
lines(v)

contour(r, filled=TRUE, nlevels=5)

## if you want a SpatVector with contour lines
template <- disagg(rast(r), 10)
rr <- resample(r, template)
rr <- floor(rr/100) * 100
v <- as.polygons(rr)
plot(v, 1, col=terrain.colors(7))

```

convHull

*Convex hull and minimal rotated rectangle***Description**

Get the convex hull or the minimal rotated rectangle of a SpatVector

Usage

```

## S4 method for signature 'SpatVector'
convHull(x, by="")

## S4 method for signature 'SpatVector'
minRect(x, by="")

```

Arguments

x	SpatVector
by	character (variable name), to make convex hulls by group

Value

SpatVector

Examples

```
p <- vect(system.file("ex/lux.shp", package="terra"))
h <- convHull(p)

hh <- convHull(p, "NAME_1")
rr <- minRect(p, "NAME_1")

plot(rr, lwd=5, border="gray")
plot(hh, "NAME_1", col=rainbow(10, alpha=.5), lwd=3, add=TRUE, plg=list(x="topright"))
lines(aggregate(p, "NAME_1"), col="blue", lty=2, lwd=2)
```

costDist

Cost distance

Description

Use a friction (cost) surface to compute the cost-distance from any cell to the border of one or more target cells.

Distances are computed by summing local distances between cells, which are connected with their neighbors in 8 directions, and assuming that the path has to go through the centers of one of the neighboring raster cells.

Distances are multiplied with the friction, thus to get the cost-distance, the friction surface must express the cost per unit distance (speed) of travel.

Usage

```
## S4 method for signature 'SpatRaster'
costDist(x, target=0, scale=1, maxiter=50, filename="", ...)
```

Arguments

x	SpatRaster
target	numeric. value of the target cells (where to compute cost-distance to)
scale	numeric. Scale factor. The cost distance is divided by this number
maxiter	numeric. The maximum number of iterations. Increase this number if you get the warning that costDistance did not converge
filename	character. output filename (optional)
...	additional arguments as for writeRaster

Value

SpatRaster

See Also

[gridDist](#), [distance](#)

Examples

```
r <- rast(ncols=5, nrows=5, crs="+proj=utm +zone=1 +datum=WGS84",
xmin=0, xmax=5, ymin=0, ymax=5, vals=1)
r[13] <- 0
d <- costDist(r)
plot(d)
text(d, digits=1)

r <- rast(ncols=10, nrows=10, xmin=0, xmax=10, ymin=0, ymax=10,
vals=10, crs="+proj=utm +zone=1 +datum=WGS84")
r[5, 1] <- -10
r[2:3, 1] <- r[1, 2:4] <- r[2, 5] <- 0
r[3, 6] <- r[2, 7] <- r[1, 8:9] <- 0
r[6, 6:10] <- NA
r[6:9, 6] <- NA

d <- costDist(r, -10)
plot(d)
text(d, digits=1, cex=.8)
```

cover

Replace values with values from another object

Description

Replace NA or other values in *SpatRaster* *x* with the values of *SpatRaster* *y*

For polygons: areas of *x* that overlap with *y* are replaced by *y* or, if *identity*=TRUE intersected with *y*.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
cover(x, y, values=NA, filename="", ...)
```

```
## S4 method for signature 'SpatVector,SpatVector'
cover(x, y, identity=FALSE, expand=TRUE)
```

Arguments

<i>x</i>	<i>SpatRaster</i> or <i>SpatVector</i>
<i>y</i>	Same as <i>x</i>
<i>values</i>	numeric. The cell values in <i>x</i> to be replaced by the values in <i>y</i>
<i>filename</i>	character. Output filename
<i>...</i>	additional arguments for writing files as in writeRaster
<i>identity</i>	logical. If TRUE overlapping areas are intersected rather than replaced
<i>expand</i>	logical. Should parts of <i>y</i> that are outside of <i>x</i> be included?

Value

SpatRaster

Examples

```

r1 <- r2 <- rast(ncols=36, nrows=18)
values(r1) <- 1:ncell(r1)
values(r2) <- runif(ncell(r2))
r2 <- classify(r2, cbind(-Inf, 0.5, NA))
r3 <- cover(r2, r1)

p <- vect(system.file("ex/lux.shp", package="terra"))
e <- as.polygons(ext(6, 6.4, 49.75, 50))
values(e) <- data.frame(y=10)

cv <- cover(p, e)
plot(cv, col=rainbow(12))
ci <- cover(p, e, identity=TRUE)
lines(e, lwd=3)

plot(ci, col=rainbow(12))
lines(e, lwd=3)

```

crds

*Get the coordinates of SpatVector geometries or SpatRaster cells***Description**

Get the coordinates of a SpatVector or SpatRaster cells. A matrix or data.frame of the x (longitude) and y (latitude) coordinates is returned.

Usage

```

## S4 method for signature 'SpatVector'
crds(x, df=FALSE, list=FALSE)

## S4 method for signature 'SpatRaster'
crds(x, df=FALSE, na.rm=TRUE)

```

Arguments

x	SpatRaster or SpatVector
df	logical. If TRUE a data.frame is returned instead of a matrix
list	logical. If TRUE a list is returned instead of a matrix
na.rm	logical. If TRUE cells that are NA are excluded

Value

matrix or data.frame

See Also

[geom](#) returns the complete structure of SpatVector geometries. For SpatRaster see [xyFromCell](#)

Examples

```
x1 <- rbind(c(-175,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
x3 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x4 <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))
z <- rbind(cbind(object=1, part=1, x1), cbind(object=2, part=1, x2),
          cbind(object=3, part=1, x3), cbind(object=3, part=2, x4))
colnames(z)[3:4] <- c('x', 'y')
z <- cbind(z, hole=0)
z[(z[, "object"]==3 & z[, "part"]==2), "hole"] <- 1

p <- vect(z, "polygons")
crds(p)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
g <- crds(v)
head(g)
```

crop

Cut out a geographic subset

Description

Cut out a part of a SpatRaster with a SpatExtent, or another object from which an extent can be obtained.

You can only crop rectangular areas, but see [mask](#) for setting cell values within SpatRaster to NA. Also note that the SpatRaster returned may not have the exactly the same extent as the SpatExtent supplied because you can only select entire cells (rows and columns), and you cannot add new areas. See methods like [resample](#) and [disagg](#) to force SpatRasters to align and [extend](#) to add rows and/or columns.

You can crop a SpatVector with another SpatVector. If these are not polygons, the minimum convex hull is used). Unlike with [intersect](#) the geometries and attributes of y are not transferred to the output. You can also crop a SpatVector with a rectangle (SpatRaster, SpatExtent).

Usage

```
## S4 method for signature 'SpatRaster'
crop(x, y, snap="near", mask=FALSE, touches=TRUE, extend=FALSE, filename="", ...)

## S4 method for signature 'SpatRasterDataset'
crop(x, y, snap="near", extend=FALSE)

## S4 method for signature 'SpatRasterCollection'
crop(x, y, snap="near", extend=FALSE)

## S4 method for signature 'SpatVector'
crop(x, y, ext=FALSE)

## S4 method for signature 'SpatGraticule'
crop(x, y)
```

Arguments

x	SpatRaster or SpatVector
y	SpatRaster, SpatVector, SpatExtent, or any other object that has a SpatExtent (ext returns a SpatExtent)
snap	character. One of "near", "in", or "out". Used to align y to the geometry of x
mask	logical. Should y be used to mask? Only used if y is a SpatVector, SpatRaster or sf
touches	logical. If TRUE and mask=TRUE, all cells touched by lines or polygons will be masked, not just those on the line render path, or whose center point is within the polygon
extend	logical. Should rows and/or columns be added if y is beyond the extent of x? Also see extend
filename	character. Output filename
...	additional arguments for writing files as in writeRaster
ext	logical. Use the extent of y instead of y. This also changes the behavior when y is an extent in two ways: (1) points that are on the extent boundary are removed and (2) lon/lat extents that go beyond -180 or 180 degrees longitude are wrapped around the earth to include areas at the other end of the dateline

Value

SpatRaster

See Also

[intersect](#), [extend](#)

See [window](#) for a virtual and sometimes more efficient way to crop a dataset.

Examples

```
r <- rast(xmin=0, xmax=10, ymin=0, ymax=10, nrows=25, ncols=25)
values(r) <- 1:ncell(r)
e <- ext(-5, 5, -5, 5)
rc <- crop(r, e)

# crop and mask
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
cm <- crop(r, v[9:12,], mask=TRUE)
plot(cm)
lines(v)

# crop vector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
e <- ext(6.15, 6.3, 49.7, 49.8)
x <- crop(v, e)
plot(x, "NAME_1")
```

crosstab	<i>Cross-tabulate</i>
----------	-----------------------

Description

Cross-tabulate the layers of a SpatRaster to create a contingency table.

Usage

```
## S4 method for signature 'SpatRaster,missing'
crosstab(x, digits=0, long=FALSE, useNA=FALSE)
```

Arguments

x	SpatRaster
digits	integer. The number of digits for rounding the values before cross-tabulation
long	logical. If TRUE the results are returned in 'long' format data.frame instead of a table
useNA	logical, indicting if the table should includes counts of NA values

Value

A table or data.frame

See Also

[freq](#), [zonal](#)

Examples

```

r <- s <- rast(nc=5, nr=5)
set.seed(1)
values(r) <- runif(ncell(r)) * 2
values(s) <- runif(ncell(r)) * 3
x <- c(r, s)

crosstab(x)

rs <- r/s
r[1:5] <- NA
s[20:25] <- NA
x <- c(r, s, rs)
crosstab(x, useNA=TRUE, long=TRUE)

```

crs

Get or set a coordinate reference system

Description

Get or set the coordinate reference system (CRS), also referred to as a "projection", of a `SpatRaster` or `SpatVector`.

Setting a new CRS does not change the data itself, it just changes the label. So you should only set the CRS of a dataset (if it does not come with one) to what it *is*, not to what you would *like* it to be. See [project](#) to *transform* an object from one CRS to another.

Usage

```

## S4 method for signature 'SpatRaster'
crs(x, proj=FALSE, describe=FALSE, parse=FALSE)

## S4 method for signature 'SpatVector'
crs(x, proj=FALSE, describe=FALSE, parse=FALSE)

## S4 method for signature 'character'
crs(x, proj=FALSE, describe=FALSE, parse=FALSE)

## S4 replacement method for signature 'SpatRaster'
crs(x, warn=FALSE)<-value

## S4 replacement method for signature 'SpatVector'
crs(x, warn=FALSE)<-value

```

Arguments

x	<code>SpatRaster</code> or <code>SpatVector</code>
proj	logical. If TRUE the crs is returned in PROJ-string notation

describe	logical. If TRUE the name, EPSG code, and the name and extent of the area of use are returned if known
warn	logical. If TRUE, a message is printed when the object already has a non-empty crs
value	character string describing a coordinate reference system. This can be in a WKT format, as a <authority:number> code such as "EPSG:4326", or a PROJ-string format such as "+proj=utm +zone=12" (see Note)
parse	logical. If TRUE, wkt parts are parsed into a vector (each line becomes an element)

Value

character or modified SpatRaster/Vector

Note

Projections are handled by the PROJ/GDAL libraries. Recent changes in the PROJ library to improve transformations between datums have degraded the library's usability. The PROJ developers suggest to no longer use the proj-string notation to define a CRS, but use the WKT2 or <authority>:<code> notation instead. These alternative systems work for formally described CRSs that are in databases, but they do not cover the infinite number of CRSs that exist. It is not practical to define one's own custom CRS with WKT2. Moreover, unlike the proj-notation, these newer systems are hard to read and that leads to code that cannot be easily understood and, therefore, is more error-prone.

It is still possible to use the PROJ-string notation with one major caveat: the datum should be WGS84 (or the equivalent NAD83) – if you want to transform your data to a coordinate reference system with a different datum. Thus as long as you use WGS84, or an ellipsoid instead of a datum, you can safely use PROJ-strings to represent your CRS; including to define your own custom CRS.

You can also set the crs to "local" to get an informal coordinate system on an arbitrary Euclidean (Cartesian) plane with units in meter.

Examples

```
r <- rast()
crs(r)
crs(r, describe=TRUE, proj=TRUE)

crs(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
crs(r)

# You can use epsg codes
crs(r) <- "epsg:25831"
crs(r, describe=TRUE)$area

crs("epsg:25831", describe=TRUE)
```

`deepcopy`*Deep copy*

Description

Make a deep copy of a `SpatRaster` or `SpatVector`. This is occasionally useful when wanting to use a replacement function in a shallow copy. That is a copy that was created like this: `x <- y`. If you use a replacement function to change an object, its shallow copies also change.

Usage

```
## S4 method for signature 'SpatRaster'
deepcopy(x)
```

```
## S4 method for signature 'SpatVector'
deepcopy(x)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
----------------	--

Value

Same as `x`

Examples

```
r <- rast(ncols=10, nrows=10, nl=3)
tm <- as.Date("2001-05-03") + 1:3
time(r) <- tm
time(r)
x <- r
time(x) <- tm + 365
time(x)
time(r)

y <- deepcopy(r)
time(y) <- tm - 365
time(y)
time(r)

# or make a new object like this
z <- rast(r)
time(z) <- tm
time(z)
time(r)
```

densify	<i>Add additional nodes to lines or polygons</i>
---------	--

Description

Add additional nodes to lines or polygons. This can be useful to do prior to using project such that the path does not change too much.

Usage

```
## S4 method for signature 'SpatVector'
densify(x, interval, equalize=TRUE, flat=FALSE)
```

Arguments

x	SpatVector
interval	positive number, specifying the desired minimum distance between nodes. The unit is meter for lonlat data, and in the linear unit of the crs for planar data
equalize	logical. If TRUE, new nodes are spread at equal intervals between old nodes
flat	logical. If TRUE, the earth's curvature is ignored for lonlat data, and the distance unit is degrees, not meter

Value

SpatVector

Examples

```
v <- vect(rbind(c(-120,-20), c(-80,5), c(-40,-60), c(-120,-20)),
  type="polygons", crs="+proj=longlat")
vd <- densify(v, 200000)

p <- project(v, "+proj=robin")
pd <- project(vd, "+proj=robin")

# good
plot(pd, col="gray", border="red", lwd=10)
points(pd, col="gray")

# bad
lines(p, col="blue", lwd=3)
points(p, col="blue", cex=2)
plot(p, col="blue", alpha=.1, add=TRUE)
legend("topright", c("good", "bad"), col=c("red", "blue"), lty=1, lwd=3)

## the other way around does not work
## unless the original data was truly planar (e.g. derived from a map)
x <- densify(p, 250000)
```

```

y <- project(x, "+proj=longlat")
# bad
plot(y)
# good
lines(vd, col="red")

```

density

Density plot

Description

Create density plots of the cell values of a SpatRaster

Usage

```

## S4 method for signature 'SpatRaster'
density(x, maxcells=100000, plot=TRUE, main, ...)

```

Arguments

x	SpatRaster
maxcells	the maximum number of (randomly sampled) cells to be used for creating the plot
plot	if TRUE produce a plot, else return a density object
main	character. Caption of plot(s)
...	additional arguments passed to plot

Value

density plot (and a density object, returned invisibly if plot=TRUE)

Examples

```

logo <- rast(system.file("ex/logo.tif", package="terra"))
density(logo)

```

deprecated	<i>deprecated methods</i>
------------	---------------------------

Description

These methods are deprecated and will be removed in future versions

Usage

```
## S4 method for signature 'SpatRaster'
setCats(x, ...)

## S4 method for signature 'SpatRaster'
area(x, ...)
```

Arguments

- x object
- ... additional arguments

depth	<i>depth of SpatRaster layers</i>
-------	-----------------------------------

Description

Get or set the depth of the layers of a SpatRaster. Experimental.

Usage

```
## S4 method for signature 'SpatRaster'
depth(x)

## S4 replacement method for signature 'SpatRaster'
depth(x)<-value
```

Arguments

- x SpatRaster
- value numeric vector

Value

numeric

See Also

[time](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))

depth(s) <- 1:3
depth(s)
```

describe	<i>describe</i>
----------	-----------------

Description

Describe the properties of spatial data in a file as generated with the "GDALinfo" tool.

Usage

```
## S4 method for signature 'character'
describe(x, sds=FALSE, meta=FALSE, parse=FALSE, options="", print=FALSE, open_opt="")
```

Arguments

x	character. The name of a file with spatial data. Or a fully specified subdataset within a file such as "NETCDF:\\"AVHRR.nc\\" :NDVI"
sds	logical. If TRUE the description or metadata of the subdatasets is returned (if available)
meta	logical. Get the file level metadata instead
parse	logical. If TRUE, metadata for subdatasets is parsed into components (if meta=TRUE)
options	character. A vector of valid options (if meta=FALSE) including "json", "mm", "stats", "hist", "nogcp", "nomd", "norat", "noct", "nofl", "checksum", "proj4", "listmdd", "mdd <value>" where <value> specifies a domain or 'all', "wkt_format <value>" where value is one of 'WKT1', 'WKT2', 'WKT2_2015', or 'WKT2_2018', "sd <subdataset>" where <subdataset> is the name or identifier of a sub-dataset. See https://gdal.org/programs/gdalinfo.html . Ignored if sds=TRUE
print	logical. If TRUE, print the results
open_opt	character. Driver specific open options

Value

character (invisibly, if print=FALSE)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
describe(f)
describe(f, meta=TRUE)
#g <- describe(f, options=c("json", "nomd", "proj4"))
#head(g)
```

diff

*Lagged differences***Description**

Compute the difference between consecutive layers in a `SpatRaster`.

Usage

```
## S4 method for signature 'SpatRaster'
diff(x, lag=1, filename="", ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>lag</code>	positive integer indicating which lag to use
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster

Value

`SpatRaster`

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
d <- diff(s)
```

dimensions

Dimensions of a SpatRaster or SpatVector and related objects

Description

Get the number of rows (nrow), columns (ncol), cells (ncell), layers (nlyr), sources (nsrc), the size size (nlyr(x)*ncell(x)), or spatial resolution of a SpatRaster.

length returns the number of sub-datasets in a SpatRasterDataset or SpatVectorCollection.

For a SpatVector length(x) is the same as nrow(x).

You can also set the number of rows or columns or layers. When setting dimensions, all cell values are dropped.

Usage

```
## S4 method for signature 'SpatRaster'
ncol(x)

## S4 method for signature 'SpatRaster'
nrow(x)

## S4 method for signature 'SpatRaster'
nlyr(x)

## S4 method for signature 'SpatRaster'
ncell(x)

## S4 method for signature 'SpatRaster'
nsrc(x)

## S4 replacement method for signature 'SpatRaster,numeric'
ncol(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
nrow(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
nlyr(x)<-value

## S4 method for signature 'SpatRaster'
res(x)

## S4 replacement method for signature 'SpatRaster,numeric'
res(x)<-value

## S4 method for signature 'SpatRaster'
xres(x)
```



```
## S4 method for signature 'SpatRaster'  
yres(x)  
  
## S4 method for signature 'SpatVector'  
ncol(x)  
  
## S4 method for signature 'SpatVector'  
nrow(x)  
  
## S4 method for signature 'SpatVector'  
length(x)
```

Arguments

x	SpatRaster or SpatVector or related objects
value	For ncol and nrow: positive integer. For res: one or two positive numbers

Value

integer

See Also

[ext](#)

Examples

```
r <- rast()  
ncol(r)  
nrow(r)  
nlyr(r)  
dim(r)  
nsrc(r)  
ncell(r)  
  
rr <- c(r,r)  
nlyr(rr)  
nsrc(rr)  
ncell(rr)  
  
nrow(r) <- 18  
ncol(r) <- 36  
# equivalent to  
dim(r) <- c(18, 36)  
  
dim(r)  
dim(r) <- c(10, 10, 5)  
dim(r)
```

```

xres(r)
yres(r)
res(r)

res(r) <- 1/120
# different xres and yres
res(r) <- c(1/120, 1/60)

```

direction

Direction

Description

The direction (azimuth) to or from the nearest cell that is not NA. The direction is expressed in radians, unless you use argument degrees=TRUE.

Usage

```

## S4 method for signature 'SpatRaster'
direction(x, from=FALSE, degrees=FALSE, filename="", ...)

```

Arguments

x	SpatRaster
filename	Character. Output filename (optional)
degrees	Logical. If FALSE (the default) the unit of direction is radians.
from	Logical. Default is FALSE. If TRUE, the direction from (instead of to) the nearest cell that is not NA is returned
...	Additional arguments as for writeRaster

Value

SpatRaster

See Also

[distance](#)

Examples

```

r <- rast(ncol=36,nrow=18, crs="+proj=merc")
values(r) <- NA
r[306] <- 1
b <- direction(r, degrees=TRUE)
plot(b)

crs(r) <- "+proj=longlat"
b <- direction(r)
plot(b)

```

disagg	<i>Disaggregate raster cells or vector geometries</i>
--------	---

Description

SpatRaster: Create a **SpatRaster** with a higher resolution (smaller cells). The values in the new **SpatRaster** are the same as in the larger original cells.

SpatVector: Separate multi-objects (points, lines, polygons) into single objects; or further into segments (for lines or polygons).

Usage

```
## S4 method for signature 'SpatRaster'
disagg(x, fact, method="near", filename="", ...)
```

```
## S4 method for signature 'SpatVector'
disagg(x, segments=FALSE)
```

Arguments

<code>x</code>	SpatRaster or SpatVector
<code>fact</code>	positive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers)
<code>method</code>	character. Either "near" for nearest or "bilinear" for bilinear interpolation
<code>segments</code>	logical. Should (poly-)lines or polygons be disaggregated into their line-segments?
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[aggregate](#), [resample](#)

Examples

```
r <- rast(ncols=10, nrows=10)
rd <- disagg(r, fact=c(10, 2))
ncol(rd)
nrow(rd)
values(r) <- 1:ncell(r)
rd <- disagg(r, fact=c(4, 2))
```

distance	<i>Geographic distance</i>
----------	----------------------------

Description

If *x* is a **SpatRaster**:

If *y* is missing this method computes the distance, for all cells that are NA in SpatRaster *x* to the nearest cell that is not NA (or other values, see arguments "target" and "exclude").

If *y* is a numeric value, the cells with that value are ignored. That is, distance to or from these cells is not computed (only if `grid=FALSE`).

If *y* is a SpatVector, the distance to that SpatVector is computed for all cells. For lines and polygons this is done after rasterization; and only the overlapping areas of the vector and raster are considered (for now).

The distance is always expressed in meter if the coordinate reference system is longitude/latitude, and in map units otherwise. Map units are typically meter, but inspect `crs(x)` if in doubt.

Results are more precise, sometimes much more precise, when using longitude/latitude rather than a planar coordinate reference system, as these distort distance.

If *x* is a **SpatVector**:

If *y* is missing, a distance matrix between all object in *x* is computed. An distance matrix object of class "dist" is returned.

If *y* is a SpatVector the geographic distance between all objects is computed (and a matrix is returned). If both sets have the same number of points, and `pairwise=TRUE`, the distance between each pair of objects is computed, and a vector is returned.

The distance is always expressed in meter, except when the coordinate reference system is longitude/latitude AND one of the SpatVector(s) consists of lines or polygons. In that case the distance is in degrees, and thus not very useful (this will be fixed soon). Otherwise, results are more precise, sometimes much more precise, when using longitude/latitude rather than a planar coordinate reference system, as these distort distance.

If *x* is a **matrix**:

x should consist of two columns, the first with "x" (or longitude) and the second with "y" coordinates (or latitude). If *y* is also a matrix, the distance between each points in *x* and all points in *y* is computed, unless `pairwise=TRUE`

If *y* is missing, the distance between each points in *x* with all other points in *x* is computed, unless `sequential=TRUE`

Usage

```
## S4 method for signature 'SpatRaster,missing'
distance(x, y, target=NA, exclude=NULL, unit="m", haversine=TRUE, filename="", ...)
```

```
## S4 method for signature 'SpatRaster,SpatVector'
distance(x, y, unit="m", rasterize=FALSE, haversine=TRUE, filename="", ...)
```

```
## S4 method for signature 'SpatVector,ANY'
distance(x, y, sequential=FALSE, pairs=FALSE, symmetrical=TRUE, unit="m")

## S4 method for signature 'SpatVector,SpatVector'
distance(x, y, pairwise=FALSE, unit="m")

## S4 method for signature 'matrix,matrix'
distance(x, y, lonlat, pairwise=FALSE)

## S4 method for signature 'matrix,missing'
distance(x, y, lonlat, sequential=FALSE, pairs=FALSE, symmetrical=TRUE)
```

Arguments

x	SpatRaster, SpatVector, or two-column matrix with coordinates (x,y) or (lon,lat)
y	missing, numeric, SpatVector, or two-column matrix
target	numeric. The value of the cells for which distances to cells that are not NA should be computed
exclude	numeric. The value of the cells that should not be considered for computing distances
unit	character. Can be either "m" or "km"
haversine	logical. Use the haversine formula for lon/lat data use the haversine formula? If FALSE, the more precise but slower method of Karney (2003) is used
rasterize	logical. If TRUE distance is computed from the cells covered by the geometries after rasterization. This can be much faster in some cases
filename	character. Output filename
...	additional arguments for writing files as in writeRaster
sequential	logical. If TRUE, the distance between sequential geometries is returned
pairwise	logical. If TRUE and if x and y have the same size (number of rows), the pairwise distances are returned instead of the distances between all elements
lonlat	logical. If TRUE the coordinates are interpreted as angular (longitude/latitude). If FALSE they are interpreted as planar
pairs	logical. If TRUE a "from", "to", "distance" matrix is returned
symmetrical	logical. If TRUE and pairs=TRUE, the distance between a pair is only included once. The distance between geometry 1 and 3 is included, but the (same) distance between 3 and 1 is not

Value

SpatRaster or numeric or matrix or distance matrix (object of class "dist")

Note

The distance unit is in meters.

A distance matrix can be coerced into a matrix with `as.matrix`

References

Karney, C.F.F., 2013. Algorithms for geodesics, J. Geodesy 87: 43-55. doi:10.1007/s00190-012-0578-z.

Examples

```
#lonlat
r <- rast(ncols=36, nrows=18, crs="+proj=longlat +datum=WGS84")
r[500] <- 1
d <- distance(r)
plot(d / 100000)

#planar
rr <- rast(ncols=36, nrows=18, crs="+proj=utm +zone=1 +datum=WGS84")
rr[500] <- 1
d <- distance(rr)

rr[3:10, 3:10] <- 99
e <- distance(rr, exclude=99)

p1 <- vect(rbind(c(0,0), c(90,30), c(-90,-30)), crs="+proj=longlat +datum=WGS84")
dp <- distance(r, p1)

d <- distance(p1)
d
as.matrix(d)

p2 <- vect(rbind(c(30,-30), c(25,40), c(-9,-3)), crs="+proj=longlat +datum=WGS84")
dd <- distance(p1, p2)
dd
pd <- distance(p1, p2, pairwise=TRUE)
pd
pd == diag(dd)

# polygons, lines
crs <- "+proj=utm +zone=1"
p1 <- vect("POLYGON ((0 0, 8 0, 8 9, 0 9, 0 0))", crs=crs)
p2 <- vect("POLYGON ((5 6, 15 6, 15 15, 5 15, 5 6))", crs=crs)
p3 <- vect("POLYGON ((2 12, 3 12, 3 13, 2 13, 2 12))", crs=crs)
p <- rbind(p1, p2, p3)
L1 <- vect("LINESTRING(1 11, 4 6, 10 6)", crs=crs)
L2 <- vect("LINESTRING(8 14, 12 10)", crs=crs)
L3 <- vect("LINESTRING(1 8, 12 14)", crs=crs)
lns <- rbind(L1, L2, L3)
pts <- vect(cbind(c(7,10,10), c(3,5,6)), crs=crs)

distance(p1,p3)
distance(p)
distance(p,pts)
distance(p,lns)
distance(pts,lns)
```

dots

*Make a dot-density map***Description**

Create the dots for a dot-density map and add these to the current map. Dot-density maps are made to display count data. For example of population counts, where each dot represents n persons. The dots are returned as a `SpatVector`. If there is an active graphics device, the dots are added to it with [points](#).

Usage

```
## S4 method for signature 'SpatVector'
dots(x, field, size, ...)
```

Arguments

<code>x</code>	<code>SpatVector</code>
<code>field</code>	character of numeric indicating field name. Or numeric vector of the same length as <code>x</code>
<code>size</code>	positive number indicating the number of cases associated with each dot
<code>...</code>	graphical arguments passed to <code>points</code>

Value

`SpatVector` (invisibly)

See Also

[plot](#), [cartogram](#), [points](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$population <- 1000*(1:12)^2
plot(v, lwd=3, col="light gray", border="white")
d <- dots(v, "population", 1000, col="red", cex=.75)
lines(v)
d
```

draw	<i>Draw a polygon, line, extent, or points</i>
------	--

Description

Draw on a plot (map) to get a `SpatVector` or `SpatExtent` object for later use. After calling the function, start clicking on the map. When you are done, press ESC. You can also preset the maximum number of clicks.

This does to work well on the default RStudio plotting device. To work around that, you can first run `dev.new(noRStudioGD = TRUE)` which will create a separate window for plotting, then use `plot()` followed by `click()` and click on the map .

Usage

```
## S4 method for signature 'character'
draw(x="extent", col="red", lwd=2, id=FALSE, n=1000, xpd=TRUE, ...)
```

Arguments

<code>x</code>	character. The type of object to draw. One of "extent", "polygon", "line", or "points"
<code>col</code>	the color to be used
<code>lwd</code>	the width of the lines to be drawn
<code>id</code>	logical. If TRUE, a numeric ID is shown on the map
<code>n</code>	the maximum number of clicks (does not apply when <code>x=="extent"</code> in which case <code>n</code> is always 2)
<code>xpd</code>	logical. If TRUE, you can draw outside the current plotting area
<code>...</code>	additional graphics arguments for drawing

Value

`SpatVector` or `SpatExtent`

See Also

[click](#)

elongate	<i>elongate lines</i>
----------	-----------------------

Description

Elongate SpatVector lines

Usage

```
## S4 method for signature 'SpatVector'
elongate(x, length=1, flat=FALSE)
```

Arguments

x	SpatVector
length	positive number indicating how much the lines should be elongated at each end. The unit is meter if the crs is lonlat and it is the same as the linear unit of the crs on other cases (also meter in most cases)
flat	logical. If TRUE, the earth's curvature is ignored for lonlat data, and the distance unit is degrees, not meter

Value

SpatVector

See Also

[buffer](#), [crop](#) and [erase](#)

Examples

```
v <- vect(cbind(c(0,1,2), c(0,0,2)), "lines", crs="lonlat")
e <- elongate(v, 100000)
plot(e)
points(e)
geom(e)
```

erase

Erase parts of a SpatVector object

Description

Erase parts of a SpatVector with another SpatVector or with a SpatExtent. You can also erase (parts of) polygons with the other polygons of the same SpatVector.

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
erase(x, y)
```

```
## S4 method for signature 'SpatVector,missing'
erase(x, sequential=TRUE)
```

```
## S4 method for signature 'SpatVector,SpatExtent'
erase(x, y)
```

Arguments

x	SpatVector
y	SpatVector or SpatExtent
sequential	logical. Should areas be erased sequentially? See Details

Details

If polygons are erased sequentially, everything that is covered by the first polygon is removed from all other polygons, then everything that is covered by (what is remaining of) the second polygon is removed, etc.

If polygons are not erased sequentially, all overlapping areas are erased and only the areas covered by a single geometry are returned.

Value

SpatVector or SpatExtent

See Also

[crop](#) and [intersect](#) for the inverse.

The equivalent for SpatRaster is [mask](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

# polygons with polygons or extent

e <- ext(5.6, 6, 49.55, 49.7)
x <- erase(v, e)

p <- vect("POLYGON ((5.8 49.8, 6 49.9, 6.15 49.8, 6 49.6, 5.8 49.8))")
y <- erase(v, p)

# lines with polygons
lms <- as.lines(rast(v, ncol=10, nrow=10))[12:22]
elm <- erase(lms, v)
plot(v)
lines(lms, col='blue', lwd=4, lty=3)
lines(elm, col='red', lwd=2)

## self-erase
h <- convHull(v[-12], "NAME_1")
he <- erase(h)
plot(h, lwd=2, border="red", lty=2)
lines(he, col="gray", lwd=3)
```

expanse	<i>Get the expanse (area) of individual polygons or for all (summed) raster cells</i>
---------	---

Description

Compute the area covered by polygons or for all raster cells that are not NA.

This method computes areas for longitude/latitude rasters, as the size of the cells is constant in degrees, but not in square meters. But it can also be important if the coordinate reference system is planar, but not equal-area.

For vector data, the best way to compute area is to use the longitude/latitude CRS. This is contrary to (erroneous) popular belief that suggest that you should use a planar coordinate reference system. This is done automatically, if `transform=TRUE`.

Usage

```
## S4 method for signature 'SpatRaster'
expanse(x, unit="m", transform=TRUE, byValue=FALSE, zones=NULL, wide=FALSE)

## S4 method for signature 'SpatVector'
expanse(x, unit="m", transform=TRUE)
```

Arguments

x	SpatRaster or SpatVector
unit	character. Output unit of area. One of "m", "km", or "ha"
transform	logical. If TRUE, planar CRS are transformed to lon/lat for accuracy
byValue	logical. If TRUE, the area for each unique cell value is returned
zones	NULL or SpatRaster with the same geometry identifying zones in x
wide	logical. Should the results be in "wide" rather than "long" format?

Value

SpatRaster: data.frame with at least two columns ("layer" and "area") and possibly also "value" (if byValue is TRUE, and "zone" (if zones is TRUE). If x has no values, the total area of all cells is returned. Otherwise, the area of all cells that are not NA is returned.

SpatVector: numeric (one value for each (multi-) polygon geometry).

See Also

[cellSize](#) for a the size of individual cells of a raster, that can be summed with [global](#) or with [zonal](#) to get the area for different zones.

Examples

```
### SpatRaster
r <- rast(nrows=18, ncols=36)
v <- 1:ncell(r)
v[200:400] <- NA
values(r) <- v

# summed area in km2
expanse(r, unit="km")

# all cells
expanse(rast(r), unit="km")

r <- rast(ncols=90, nrows=45, ymin=-80, ymax=80)
m <- project(r, "+proj=merc")

expanse(m, unit="km")
expanse(m, unit="km", transform=FALSE)

m2 <- c(m, m)
values(m2) <- cbind(c(1,2,NA,NA), c(11:14))
expanse(m2, unit="km", byValue=TRUE, wide=TRUE)

v <- vect(system.file("ex/lux.shp", package="terra"))
r <- rast(system.file("ex/elev.tif", package="terra"))
r <- round((r-50)/100)
```

```

levels(r) <- data.frame(id=1:5, name=c("forest", "water", "urban", "crops", "grass"))
expanse(r, byValue=TRUE)

g <- rasterize(v, r, "NAME_1")
expanse(r, byValue=TRUE, zones=g, wide=TRUE)

### SpatVector
v <- vect(system.file("ex/lux.shp", package="terra"))

a <- expanse(v)
a
sum(a)

```

ext

Create, get or set a SpatExtent

Description

Get a SpatExtent of a SpatRaster, SpatVector, or other spatial objects. Or create a SpatExtent from a vector (length=4; order=xmin, xmax, ymin, ymax)

You can set the extent of a SpatRaster, but you cannot set the extent of a SpatVector (see [rescale](#) for that). See [set.ext](#) to set the extent in place.

Usage

```

## S4 method for signature 'SpatRaster'
ext(x, cells=NULL)

## S4 method for signature 'SpatVector'
ext(x)

## S4 method for signature 'numeric'
ext(x, ..., xy=FALSE)

## S4 replacement method for signature 'SpatRaster,SpatExtent'
ext(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
ext(x)<-value

```

Arguments

x	SpatRaster
cells	positive integer (cell) numbers to subset the extent to area covered by these cells
value	SpatExtent, or numeric vector of length four (xmin, xmax, ymin, ymax)

...	if x is a single numeric value, additional numeric values for xmax, ymin, and ymax
xy	logical. Set this to TRUE to indicate that coordinates are in (xmin, ymin, xmax, ymax) order, instead of in the terra standard order of (xmin, xmax, ymin, ymax)

Value

A [SpatExtent](#) object.

Examples

```
r <- rast()
e <- ext(r)
as.vector(e)
as.character(e)

ext(r) <- c(0, 2.5, 0, 1.5)
r
er <- ext(r)

round(er)
# go "in"
floor(er)
# go "out"
ceiling(er)

ext(r) <- e
```

extend	<i>Extend</i>
--------	---------------

Description

Enlarge the spatial extent of a [SpatRaster](#). See [crop](#) if you (also) want to remove rows or columns.
You can also enlarge a [SpatExtent](#) with this method, or with algebraic notation (see examples)

Usage

```
## S4 method for signature 'SpatRaster'
extend(x, y, snap="near", fill=NA, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatExtent'
extend(x, y)
```

Arguments

x	SpatRaster or SpatExtent
y	If x is a SpatRaster, y should be a SpatExtent, or an object from which it can be extracted (such as SpatRaster and SpatVector objects). Alternatively, you can provide two positive integers indicating the number of rows and columns that need to be added at each side (or a single positive integer when the number of rows and columns is equal) If x is a SpatExtent, y should be a numeric vector of 1, 2, or 4 elements
snap	character. One of "near", "in", or "out". Used to align y to the geometry of x
fill	numeric. The value used to for the new raster cells
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
...	additional arguments for writing files as in writeRaster

Value

SpatRaster or SpatExtent

See Also

[crop](#), [merge](#), [ext](#)

Examples

```
r <- rast(xmin=-150, xmax=-120, ymin=30, ymax=60, ncols=36, nrows=18)
values(r) <- 1:ncell(r)
e <- ext(-180, -100, 40, 70)
re <- extend(r, e)

# extend with a number of rows and columns (at each side)
re2 <- extend(r, c(2,10))

# SpatExtent
e <- ext(r)
e
extend(e, 10)
extend(e, c(10, -10, 0, 20))
```

Description

Extract values from a `SpatRaster` for a set of locations. The locations can be a `SpatVector` (points, lines, polygons), a matrix with (x, y) or (longitude, latitude – in that order!) coordinates, or a vector with cell numbers.

When argument `y` is a `SpatVector` the first column has the ID (record number) of the `SpatVector` used (unless you set `ID=FALSE`).

Alternatively, you can use [zonal](#) after using [rasterize](#) with a `SpatVector` (this may be more efficient in some cases).

Usage

```
## S4 method for signature 'SpatRaster,SpatVector'
extract(x, y, fun=NULL, method="simple", cells=FALSE, xy=FALSE,
        ID=TRUE, weights=FALSE, exact=FALSE, touches=is.lines(y),
        layer=NULL, bind=FALSE, raw=FALSE, ...)

## S4 method for signature 'SpatRaster,SpatExtent'
extract(x, y, cells=FALSE, xy=FALSE)

## S4 method for signature 'SpatRaster,matrix'
extract(x, y, cells=FALSE, method="simple")

## S4 method for signature 'SpatRaster,numeric'
extract(x, y, xy=FALSE)

## S4 method for signature 'SpatVector,SpatVector'
extract(x, y)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code> of polygons
<code>y</code>	<code>SpatVector</code> (points, lines, or polygons). Alternatively, for points, a 2-column matrix or <code>data.frame</code> (x, y) or (lon, lat). Or a vector with cell numbers
<code>fun</code>	function to summarize the data by geometry. You can use <code>fun=table</code> to tabulate raster values for each line or polygon geometry. If <code>weights=TRUE</code> or <code>exact=TRUE</code> only mean, sum, min, max and table are accepted)
<code>method</code>	character. method for extracting values with points ("simple" or "bilinear"). With "simple" values for the cell a point falls in are returned. With "bilinear" the returned values are interpolated from the values of the four nearest raster cells
<code>cells</code>	logical. If TRUE the cell numbers are also returned, unless <code>fun</code> is not NULL. Also see cells
<code>xy</code>	logical. If TRUE the coordinates of the cells are also returned, unless <code>fun</code> is not NULL. Also see xyFromCell
<code>ID</code>	logical. Should an ID column be added? If so, the first column returned has the IDs (record numbers) of input <code>SpatVector</code> <code>y</code>

weights	logical. If TRUE and y has polygons, the approximate fraction of each cell that is covered is returned as well, for example to compute a weighted mean
exact	logical. If TRUE and y has polygons, the exact fraction of each cell that is covered is returned as well, for example to compute a weighted mean
touches	logical. If TRUE, values for all cells touched by lines or polygons are extracted, not just those on the line render path, or whose center point is within the polygon. Not relevant for points; and always considered TRUE when weights=TRUE or exact=TRUE
layer	character or numeric to select the layer to extract from for each geometry. If layer is a character it can be a name in y or a vector of layer names. If it is numeric, it must be integer values between 1 and nlyr(x)
bind	logical. If TRUE, a SpatVector is returned consisting of the input SpatVector y and the cbind-ed extracted values
raw	logical. If TRUE, a matrix is returned with the "raw" numeric cell values. If FALSE, a data.frame is returned and the cell values are transformed to factor, logical, or integer values, where appropriate
...	additional arguments to fun if y is a SpatVector. For example na.rm=TRUE. Or arguments passed to the SpatRaster, SpatVector method if y is a matrix (such as the method and cells arguments)

Value

data.frame, matrix or SpatVector

See Also

[values](#), [zonal](#)

Examples

```
r <- rast(ncols=5, nrows=5, xmin=0, xmax=5, ymin=0, ymax=5)
values(r) <- 1:25
xy <- rbind(c(0.5,0.5), c(2.5,2.5))
p <- vect(xy, crs="+proj=longlat +datum=WGS84")

extract(r, xy)
extract(r, p)

r[1,]
r[5]
r[,5]

r[c(0:2, 99:101)]

f <- system.file("ex/meuse.tif", package="terra")
r <- rast(f)

xy <- cbind(179000, 330000)
xy <- rbind(xy-100, xy, xy+1000)
```

```

extract(r, xy)

p <- vect(xy)
g <- geom(p)
g

extract(r, p)

x <- r + 10
extract(x, p)

i <- cellFromXY(r, xy)
x[i]
r[i]

y <- c(x,x*2,x*3)
y[i]

## extract with a polygon
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v <- v[1:2,]
z <- rast(v, resolution=.1, names="test")
values(z) <- 1:ncell(z)

rf <- system.file("ex/elev.tif", package="terra")
x <- rast(rf)
extract(x, v, mean, na.rm=TRUE)

e <- extract(z, v, ID=TRUE)
e
tapply(e[,2], e[,1], mean, na.rm=TRUE)

x <- c(z, z*2, z/3)
names(x) <- letters[1:3]

e <- extract(x, v, ID=TRUE)
de <- data.frame(e)
aggregate(de[,2:4], de[,1,drop=FALSE], mean)

```

extremes

Get or compute the minimum and maximum cell values

Description

The minimum and maximum value of a SpatRaster are returned or computed (from a file on disk if necessary) and stored in the object.

Usage

```
## S4 method for signature 'SpatRaster'
minmax(x, compute=FALSE)
## S4 method for signature 'SpatRaster'
hasMinMax(x)
## S4 method for signature 'SpatRaster'
setMinMax(x, force=FALSE)
```

Arguments

x	SpatRaster
compute	logical. If TRUE min and max values are computed if they are not available
force	logical. If TRUE min and max values are recomputed even if already available

Value

minmax: numeric matrix of minimum and maximum cell values by layer

hasMinMax: logical indicating whether the min and max values are available.

setMinMax: nothing. Used for the side-effect of computing the minimum and maximum values of a SpatRaster

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
minmax(r)
```

factors

Categorical rasters

Description

A SpatRaster layer can represent a categorical variable (factor). Like [factors](#), SpatRaster categories are stored as integers that have an associated label.

The categories can be inspected with `levels` and `cats`. They are represented by a `data.frame` that must have two or more columns, the first one identifying the (integer) cell values and the other column(s) providing the category labels.

If there are multiple columns with categories, you can set the "active" category to choose the one you want to use.

`cats` returns the entire `data.frame`, whereas `levels` only return two columns: the index and the active category.

To set categories for the first layer of a SpatRaster, you can provide `levels<-` with a `data.frame` or a list with a `data.frame`. To set categories for multiple layers you can provide `levels<-` with a list with one element (that either has a `data.frame` or is `NULL`) for each layer. Use `categories` to set the categories for a specific layer or specific layers.

`droplevels` removes categories that are not used (declared but not present as values in the raster) if `levels=NULL`.

Usage

```
## S4 method for signature 'SpatRaster'
levels(x)

## S4 replacement method for signature 'SpatRaster'
levels(x)<-value

## S4 method for signature 'SpatRaster'
cats(x, layer)

## S4 method for signature 'SpatRaster'
categories(x, layer=1, value, active=1, ...)

## S4 method for signature 'SpatRaster'
droplevels(x, level=NULL, layer=1)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>layer</code>	the layer name or number (positive integer); or 0 for all layers
<code>value</code>	a data.frame (ID, category) that define the categories. Or NULL to remove them
<code>active</code>	positive integer, indicating the column in <code>value</code> to be used as the active category (zero based to skip the first column with the cell values; that is 1 is the second column in <code>value</code>)
<code>level</code>	the categories to remove for the layer specified with <code>layer</code>
<code>...</code>	additional arguments

Value

list of data.frames (`levels`, `cats`) or logical (`is.factor`)

See Also

[activeCat](#), [catalyze](#), [set.cats](#), [as.factor](#), [is.factor](#)

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10)
values(r) <- sample(3, ncell(r), replace=TRUE)
is.factor(r)

cls <- data.frame(id=1:3, cover=c("forest", "water", "urban"))
levels(r) <- cls
is.factor(r)
```

```

r

plot(r, col=c("green", "blue", "light gray"))
text(r, digits=3, cex=.75, halo=TRUE)

# raster starts at 3
x <- r + 2
is.factor(x)

# Multiple categories
d <- data.frame(id=3:5, cover=cls[,2], letters=letters[1:3], value=10:12)
levels(x) <- d
x

# get current index
activeCat(x)
# set index
activeCat(x) <- 3
activeCat(x)
activeCat(x) <- "letters"
plot(x, col=c("green", "blue", "light gray"))
text(x, digits=3, cex=.75, halo=TRUE)

r <- as.numeric(x)
r

p <- as.polygons(x)
plot(p, "letters", col=c("green", "blue", "light gray"))

```

fillHoles

Remove holes from polygons

Description

Remove the holes in SpatVector polygons. If inverse=TRUE the holes are returned (as polygons).

Usage

```
## S4 method for signature 'SpatVector'
fillHoles(x, inverse=FALSE)
```

Arguments

x	SpatVector
inverse	logical. If TRUE the holes are returned as polygons

Value

SpatVector

Examples

```
x <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
hole <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))

z <- rbind(cbind(object=1, part=1, x, hole=0),
          cbind(object=1, part=1, hole, hole=1))
colnames(z)[3:4] <- c('x', 'y')
p <- vect(z, "polygons", atts=data.frame(id=1))
p

f <- fillHoles(p)
g <- fillHoles(p, inverse=TRUE)

plot(p, lwd=16, border="gray", col="light yellow")
polys(f, border="blue", lwd=3, density=4, col="orange")
polys(g, col="white", lwd=3)
```

fillTime	<i>Fill time gaps in a SpatRaster</i>
----------	---------------------------------------

Description

Add empty layers in between existing layers such that the time step between each layer is the same. See [approximate](#) to estimate values for these layer (and other missing values)

Usage

```
## S4 method for signature 'SpatRaster'
fillTime(x, filename="", ...)
```

Arguments

- x SpatRaster
- filename character. Output filename
- ... list with named options for writing files as in [writeRaster](#)

Value

SpatRaster

See Also

[approximate](#)

Examples

```

r <- rast(system.file("ex/logo.tif", package="terra"))
s <- c(r, r)
time(s) <- as.Date("2001-01-01") + c(0:2, 5:7)
time(s)
ss <- fillTime(s)
time(ss)

a <- approximate(ss)

```

flip	<i>Flip or reverse a raster</i>
------	---------------------------------

Description

Flip the values of a `SpatRaster` by inverting the order of the rows (`vertical=TRUE`) or the columns (`vertical=FALSE`).

`rev` is the same as a horizontal *and* a vertical flip.

Usage

```

## S4 method for signature 'SpatRaster'
flip(x, direction="vertical", filename="", ...)

## S4 method for signature 'SpatVector'
flip(x, direction="vertical")

## S4 method for signature 'SpatRaster'
rev(x)

```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>direction</code>	character. Should (partially) match "vertical" to flip by rows, or "horizontal" to flip by columns
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster

Value

`SpatRaster`

See Also

[trans](#), [rotate](#)

Examples

```
r <- rast(nrow=18, ncol=36)
m <- matrix(1:ncell(r), nrow=18)
values(r) <- as.vector(t(m))
rx <- flip(r, direction="h")

values(r) <- as.vector(m)
ry <- flip(r, direction="v")

v <- rev(r)
```

focal	<i>Focal values</i>
-------	---------------------

Description

Calculate focal ("moving window") values for each cell.

Usage

```
## S4 method for signature 'SpatRaster'
focal(x, w=3, fun="sum", ..., na.policy="all", fillvalue=NA,
expand=FALSE, silent=TRUE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster
w	window. The window can be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See Details.
fun	function that takes multiple numbers, and returns a numeric vector (one or multiple numbers). For example mean, modal, min or max
...	additional arguments passed to fun such as na.rm
na.policy	character. Can be used to determine the cells of x for which focal values should be computed. Must be one of "all" (compute for all cells), "only" (only for cells that are NA) or "omit" (skip cells that are NA). Note that the value of this argument does not affect which cells around each focal cell are included in the computations (use na.rm=TRUE to ignore cells that are NA for that)
fillvalue	numeric. The value of the cells in the virtual rows and columns outside of the raster
expand	logical. If TRUE The value of the cells in the virtual rows and columns outside of the raster are set to be the same as the value on the border. Only available for "build-in" funs such as mean, sum, min and max
silent	logical. If TRUE error messages are printed that may occur when trying fun to determine the length of the returned value. This can be useful in debugging a fun that does not work

filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	additional arguments for writing files as in writeRaster

Details

focal The window used must have odd dimensions. If you need even sides, you can use a matrix and add a column or row with weights of zero.

Window values are typically 0 or 1, or a value between 0 and 1 if you are using a rectangular area and/or the "sum" function. They can also be NA; these are ignored in the computation. That can be useful to compute, for example, the minimum or maximum value for a non-rectangular area.

The "mean" function is a special case, as zero weights are ignored automatically.

The "sum" function returns NA if all focal cells are NA and `na.rm=TRUE`. R would normally return a zero in these cases. See the difference between `focal(x, fun=sum, na.rm=TRUE)` and `focal(x, fun=\(i) sum(i, na.rm=TRUE))`

Example weight matrices

Laplacian filter: `filter=matrix(c(0,1,0,1,-4,1,0,1,0), nrow=3)`

Sobel filters (for edge detection):

`fx=matrix(c(-1,-2,-1,0,0,0,1,2,1), nrow=3)`

`fy=matrix(c(1,0,-1,2,0,-2,1,0,-1), nrow=3)`

Value

SpatRaster

See Also

[focalMat](#), [focalValues](#), [focal3D](#), [focalCor](#), [focalReg](#), [focalCpp](#)

Examples

```
r <- rast(ncols=10, nrows=10, ext(0, 10, 0, 10))
values(r) <- 1:ncell(r)

f <- focal(r, w=3, fun=function(x, ...) quantile(x, c(.25, .5, .75), ...), na.rm=TRUE)

f <- focal(r, w=3, fun="mean")

# the following two statements are equivalent:
a <- focal(r, w=matrix(1/9, nc=3, nr=3))
b <- focal(r, w=3, fun=mean, na.rm=FALSE)

# but this is different
d <- focal(r, w=3, fun=mean, na.rm=TRUE)

## illustrating the effect of different
## combinations of na.rm and na.policy
v <- vect(system.file("ex/lux.shp", package="terra"))
```

```

r <- rast(system.file("ex/elev.tif", package="terra"))
r[45:50, 45:50] <- NA

# also try "mean" or "min"
f <- "sum"
# na.rm=FALSE
plot(focal(r, 5, f) , fun=lines(v))

# na.rm=TRUE
plot(focal(r, 5, f, na.rm=TRUE), fun=lines(v))

# only change cells that are NA
plot(focal(r, 5, f, na.policy="only", na.rm=TRUE), fun=lines(v))

# do not change cells that are NA
plot(focal(r, 5, f, na.policy="omit", na.rm=TRUE), fun=lines(v))

# does not do anything
# focal(r, 5, f, na.policy="only", na.rm=FALSE)

```

focal3D

Three-dimensional focal values

Description

Calculate focal ("moving window") values for the three-dimensional neighborhood (window) of focal cells. See [focal](#) for two-dimensional focal computation.

Usage

```

## S4 method for signature 'SpatRaster'
focal3D(x, w=3, fun=mean, ..., na.policy="all", fillvalue=NA, pad=FALSE,
padvalue=fillvalue, expand=FALSE, silent=TRUE,
filename="", overwrite=FALSE, wopt=list())

```

Arguments

x	SpatRaster
w	window. A rectangular prism (cuboid) defined by three numbers or by a three-dimensional array. The values are used as weights, and are usually zero, one, NA, or fractions. The window used must have odd dimensions. If you desire to use even sides, you can use an array, and pad the values with rows and/or columns that contain only NAs.
fun	function that takes multiple numbers, and returns one or multiple numbers for each focal area. For example mean, modal, min or max
...	additional arguments passed to fun such as na.rm

na.policy	character. Can be used to determine the cells of x, in the central layer, for which focal values should be computed. Must be one of "all" (compute for all cells), "only" (only for cells that are NA) or "omit" (skip cells that are NA). Note that the value of this argument does not affect which cells around each focal cell are included in the computations (use na.rm=TRUE to ignore cells that are NA in the computation of the focal value)
fillvalue	numeric. The value of the cells in the virtual rows and columns outside of the raster
pad	logical. Add virtual layers before the first and after the last layer
padvalue	numeric. The value of the cells in the virtual layers
expand	logical. Add virtual layers before the first or after the last layer that are the same as the first or last layers. If TRUE, arguments pad and padvalue are ignored
silent	logical. If TRUE error messages are printed that may occur when trying fun to determine the length of the returned value. This can be useful in debugging a function passed to fun that does not work
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also[focal](#)**Examples**

```

r <- rast(system.file("ex/logo.tif", package="terra"))
x <- focal3D(r, c(5,5,3), na.rm=TRUE)

a <- array(c(0,1,0,1,1,1,0,1,0, rep(1,9), 0,1,0,1,1,1,0,1,0), c(3,3,3))
a[a==0] <- NA
z <- focal3D(r, a, na.rm=TRUE)

```

focalCpp

*Compute focal values with an iterating C++ function***Description**

Calculate focal values with a C++ function that iterates over cells to speed up computations by avoiding an R loop (with apply).

See [focal](#) for an easier to use method.

Usage

```
## S4 method for signature 'SpatRaster'
focalCpp(x, w=3, fun, ..., fillvalue=NA,
silent=TRUE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster
w	window. The window can be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See the Details section in focal
fun	cppFunction that iterates over cells. For C++ functions that operate on a single focal window, or for R functions use focal instead. The function must have at least three arguments. The first argument can have any name, but it must be a <code>Rcpp::NumericVector</code> , <code>Rcpp::IntegerVector</code> or a <code>std::vector<double></code> . This is the container that receives the focal values. The other two arguments <code>ni</code> and <code>wi</code> must be of type <code>size_t</code> . <code>ni</code> represents the number of cells and <code>nw</code> represents the size of (number of elements in) the window
...	additional arguments to fun
fillvalue	numeric. The value of the cells in the virtual rows and columns outside of the raster
silent	logical. If TRUE error messages are printed that may occur when trying fun to determine the length of the returned value. This can be useful in debugging a fun that does not work
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[focal](#), [focalValues](#)

Examples

```
## Not run:
library(Rcpp)
cppFunction(
'NumericVector sum_and_multiply(NumericVector x, double m, size_t ni, size_t nw) {
NumericVector out(ni);
// loop over cells
size_t start = 0;
for (size_t i=0; i<ni; i++) {
size_t end = start + nw;
// compute something for a window
```

```

double v = 0;
// loop over the values of a window
for (size_t j=start; j<end; j++) {
  v += x[j];
}
out[i] = v * m;
start = end;
}
return out;
}'
)

nr <- nc <- 10
r <- rast(ncols=nc, nrows=nr, ext= c(0, nc, 0, nr))
values(r) <- 1:ncell(r)

raw <- focalCpp(r, w=3, fun=sum_and_multiply, fillvalue=0, m=10)

# same as
f1 <- focal(r, w=3, fun=sum, fillvalue=0) *10
all(values(f1) == values(raw))

# and as
ffun <- function(x, m) { sum(x) * m }
f2 <- focal(r, w=3, fun=ffun, fillvalue=0, m=10)

# You can also use an R function with focalCpp but this
# is not recommended

R_sm_iter <- function(x, m, ni, nw) {
  out <- NULL
  for (i in 1:ni) {
    start <- (i-1) * nw + 1
    out[i] <- sum(x[start:(start+nw-1)]) * m
  }
  out
}

fr <- focalCpp(r, w=3, fun=R_sm_iter, fillvalue=0, m=10)

## End(Not run)

```

focalMat

*Focal weights matrix***Description**

Make a focal ("moving window") weight matrix for use in the [focal](#) function. The sum of the values adds up to one.

Usage

```
focalMat(x, d, type=c('circle', 'Gauss', 'rectangle'), fillNA=FALSE)
```

Arguments

x	SpatRaster
d	numeric. If type=circle, the radius of the circle (in units of the crs). If type=rectangle the dimension of the rectangle (one or two numbers). If type=Gauss the size of sigma, and optionally another number to determine the size of the matrix returned (default is 3*sigma)
type	character indicating the type of filter to be returned
fillNA	logical. If TRUE, zeros are set to NA such that they are ignored in the computations. Only applies to type="circle"

Value

matrix that can be used with `focal`

Examples

```
r <- rast(ncols=180, nrows=180, xmin=0)
focalMat(r, 2, "circle")

focalMat(r, c(2,3), "rect")

# Gaussian filter for square cells
gf <- focalMat(r, 1, "Gauss")
```

focalPairs	<i>Focal function across two layers</i>
------------	---

Description

Calculate values such as a correlation coefficient for focal regions in two neighboring layers. A function is applied to the first and second layer, then to the second and third layer, etc.

Usage

```
## S4 method for signature 'SpatRaster'
focalPairs(x, w=3, fun, ..., fillvalue=NA,
filename="", overwrite=FALSE, wopt=list())
```

Arguments

<code>x</code>	SpatRaster with at least two layers
<code>w</code>	numeric or matrix to define the focal window. The window can be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See the Details section in focal . Note that if a matrix with numbers other than zero or one are used, the values are used as weights. For this to work, fun must have an argument weights
<code>fun</code>	a function with at least two arguments (one for each layer). There is a built-in function "pearson" (for both the weighted and the unweighted Pearson correlation coefficient. This function has an additional argument <code>na.rm=FALSE</code>
<code>...</code>	additional arguments for fun
<code>fillvalue</code>	numeric. The value of the cells in the virtual rows and columns outside of the raster
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>wopt</code>	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[layerCor](#), [focalReg](#), [focal](#), [focal3D](#)

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
set.seed(0)
r[[1]] <- flip(r[[1]], "horizontal")
r[[2]] <- flip(r[[2]], "vertical") + init(rast(r,1), runif)
r[[3]] <- init(rast(r,1), runif)

x <- focalPairs(r, w=5, "pearson", na.rm=TRUE)
plot(x)

# suppress warning "the standard deviation is zero"
suppressWarnings(x <- focalPairs(r, w=5, "pearson", use="complete.obs"))

z <- focalPairs(r, w=9, function(x, y) mean(x) + mean(y))
```

focalReg

*Focal regression***Description**

Calculate values for a moving-window by comparing the value in one layers with the values in one to many other layers. A typical case is the computation of the coefficients for a focal linear regression model.

Usage

```
## S4 method for signature 'SpatRaster'
focalReg(x, w=3, fun="ols", ..., fillvalue=NA, filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster with at least two layers. The first is the "Y" (dependent) variable and the remainder are the "X" (independent) variables
w	numeric or matrix to define the focal window. The window can be defined as one (for a square) or two numbers (row, col); or with an odd-sized weights matrix. See the Details section in focal . Note that if a matrix with numbers other than zero or one are used, the values are used as weights. For this to work, fun must have an argument weights
fun	a function with at least two arguments (one for each layer). There is a built-in function "ols" for both the weighted and unweighted Ordinary Least Square regression. This function has an additional argument na.rm=FALSE and intercept=TRUE
...	additional arguments for fun
fillvalue	numeric. The value of the cells in the virtual rows and columns outside of the raster
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[focal](#), [focal3D](#), [focalValues](#)

Examples

```
r <- rast(ncols=10, nrows=10, ext(0, 10, 0, 10))
values(r) <- 1:ncell(r)
x <- c(r, init(r, runif) * r)
f <- focalReg(x, 3)
```

focalValues	<i>Get focal values</i>
-------------	-------------------------

Description

Get a matrix in which each row had the focal values of a cell. These are the values of a cell and a rectangular window around it.

Usage

```
## S4 method for signature 'SpatRaster'
focalValues(x, w=3, row=1, nrow=nrow(x), fill=NA)
```

Arguments

x	SpatRaster or SpatVector
w	window. The window can be defined as one (for a square) or two odd numbers (row, col); or with an odd sized matrix
row	positive integer. Row number to start from, should be between 1 and nrow(x)
nrow	positive integer. How many rows?
fill	numeric used as values for imaginary cells outside the raster

Value

matrix

Examples

```
r <- rast(ncol=4, nrow=4, crs="+proj=utm +zone=1 +datum=WGS84")
values(r) <- 1:ncell(r)
focalValues(r)
```

freq	<i>Frequency table</i>
------	------------------------

Description

Frequency table of the values of a SpatRaster. NAs are not counted unless value=NA.

You can provide a SpatVector or additional SpatRaster to define zones for which to do tabulations.

Usage

```
## S4 method for signature 'SpatRaster'
freq(x, digits=0, value=NULL, bylayer=TRUE, usenames=FALSE, zones=NULL, wide=FALSE)
```

Arguments

x	SpatRaster
digits	integer. Used for rounding the values before tabulation. Ignored if NA
value	numeric. An optional single value to only count the number of cells with that value. This value can be NA
bylayer	logical. If TRUE tabulation is done by layer
usenames	logical. If TRUE layers are identified by their names instead of their numbers. Only relevant if bylayer is TRUE
zones	SpatRaster or SpatVector to define zones for which the tabulation should be done
wide	logical. Should the results be by "wide" instead of "long"?

Value

A data.frame with 3 columns (layer, value, count) unless bylayer=FALSE in which case a data.frame with two columns is returned (value, count).

Examples

```
r <- rast(nrows=10, ncols=10)
set.seed(2)
values(r) <- sample(5, ncell(r), replace=TRUE)

freq(r)

x <- c(r, r/3)
freq(x, bylayer=FALSE)
freq(x)

freq(x, digits=1)
freq(x, digits=-1)

freq(x, value=5)
```

gaps	<i>Find gaps between polygons</i>
------	-----------------------------------

Description

Get the gaps between polygons of a SpatVector

Usage

```
## S4 method for signature 'SpatVector'
gaps(x)
```

Arguments

x SpatVector

Value

SpatVector

See Also

[sharedPaths](#), [topology](#), and [fillHoles](#) to get or remove polygon holes

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
h <- convHull(v[-12], "NAME_1")
g <- gaps(h)
```

gdal

GDAL version, supported file formats, and cache size

Description

Set the GDAL warning level or get a data.frame with the available GDAL drivers (file formats), or, if warn=NA and drivers=FALSE, you get the version numbers of one or all of the GDAL, PROJ and GEOS libraries.

GDAL is the software library that terra builds on to read and write spatial data and for some raster data processing. PROJ is used for transformation of coordinates ("projection") and GEOS is used for geometric operations with vector data.

Usage

```
gdal(warn=NA, drivers=FALSE, lib="gdal")
gdalCache(size=NA)
setGDALconfig(option, value="")
getGDALconfig(option)
```

Arguments

warn	If NA and drivers=FALSE, the version of the library specified by lib is returned. Otherwise, the value should be an integer between 1 and 4 representing the level of GDAL warnings and errors that are passed to R. 1 = warnings and errors; 2 = errors only (recoverable errors as a warning); 3 = irrecoverable errors only; 4 = ignore all errors and warnings. The default setting is 3
drivers	logical. If TRUE a data.frame with the raster and vector data formats that are available.

lib	character. "gdal", "proj", or "geos", or any other value to get the versions numbers of all three
size	numeric. The new cache size in MB
option	character. GDAL configuration option name, or a "name=value" string (in which case the value argument is ignored)
value	character. value for GDAL configuration option. Use "" to reset it to its default value

Value

character

See Also

[describe](#) for file-level metadata "GDALinfo"

Examples

```
gdal()
gdal(2)
head(gdal(drivers=TRUE))
```

geom

Get the geometry (coordinates) of a SpatVector

Description

Get the geometry of a SpatVector. If wkt=FALSE, this is a five-column matrix or data.frame: the vector object ID, the IDs for the parts of each object (e.g. five polygons that together are one spatial object), the x (longitude) and y (latitude) coordinates, and a flag indicating whether the part is a "hole" (only relevant for polygons).

If wkt=TRUE, the "well-known text" representation is returned as a character vector.

Usage

```
## S4 method for signature 'SpatVector'
geom(x, wkt=FALSE, hex=FALSE, df=FALSE, list=FALSE, xnm="x", ynm="y")
```

Arguments

x	SpatVector
wkt	logical. If TRUE the WKT geometry is returned (unless hex is also TRUE)
hex	logical. If TRUE the hexadecimal geometry is returned
df	logical. If TRUE a data.frame is returned instead of a matrix (only if wkt=FALSE, hex=FALSE, and list=FALSE)
list	logical. If TRUE a nested list is returned with data.frames of coordinates
xnm	character. If list=TRUE the "x" column name for the coordinates data.frame
ynm	character. If list=TRUE the "y" column name for the coordinates data.frame

Value

matrix, vector, data.frame, or list

See Also

[crds](#), [xyFromCell](#)

Examples

```
x1 <- rbind(c(-175,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
x3 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x4 <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))
z <- rbind(cbind(object=1, part=1, x1), cbind(object=2, part=1, x2),
          cbind(object=3, part=1, x3), cbind(object=3, part=2, x4))
colnames(z)[3:4] <- c('x', 'y')
z <- cbind(z, hole=0)
z[(z[, "object"]==3 & z[, "part"]==2), "hole"] <- 1

p <- vect(z, "polygons")
geom(p)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
g <- geom(v)
head(g)

w <- geom(v, wkt=TRUE)
substr(w, 1, 60)
```

geomtype

Geometry type of a SpatVector

Description

Get the geometry type (points, lines, or polygons) of a SpatVector or the data types of the fields (attributes, variables) of a SpatVector.

Usage

```
## S4 method for signature 'SpatVector'
geomtype(x)

## S4 method for signature 'SpatVector'
datatype(x)

## S4 method for signature 'SpatVector'
is.points(x)
```

```
## S4 method for signature 'SpatVector'
is.lines(x)

## S4 method for signature 'SpatVector'
is.polygons(x)

## S4 method for signature 'SpatRaster'
datatype(x, bylyr=TRUE)
```

Arguments

x	SpatVector or SpatRaster
bylyr	logical. If TRUE a value is returned for each layer. Otherwise, a value is returned for each data source (such as a file)

Value

character

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

geomtype(v)
is.polygons(v)
is.lines(v)
is.points(v)

names(v)
datatype(v)
```

global	<i>global statistics</i>
--------	--------------------------

Description

Compute global statistics, that is summarized values of an entire SpatRaster.

If x is very large global will fail, except when fun is one of "mean", "min", "max", "sum", "prod", "range" (min and max), "rms" (root mean square), "sd" (sample standard deviation), "std" (population standard deviation), "isNA" (number of cells that are NA), "notNA" (number of cells that are not NA).

You can compute a weighted mean or sum by providing a SpatRaster with weights.

Usage

```
## S4 method for signature 'SpatRaster'
global(x, fun="mean", weights=NULL, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>fun</code>	function to be applied to summarize the values by zone. Either as one or more of these character values: "max", "min", "mean", "sum", "range", "rms" (root mean square), "sd", "std" (population sd, using <code>n</code> rather than <code>n-1</code>), "isNA", "notNA"; or a proper R function (but these may fail for very large <code>SpatRasters</code>)
<code>...</code>	additional arguments passed on to <code>fun</code>
<code>weights</code>	<code>NULL</code> or <code>SpatRaster</code>

Value

A `data.frame` with a row for each layer

See Also

[zonal](#) for "zonal" statistics, and [app](#) or [Summary-methods](#) for "local" statistics, and [extract](#) for summarizing values for polygons. Also see [focal](#) for "focal" or "moving window" operations.

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
global(r, "sum")
global(r, "mean", na.rm=TRUE)
x <- c(r, r/10)
global(x, c("sum", "mean", "sd"), na.rm=TRUE)

global(x, function(i) min(i) / max(i))
```

graticule

Create a graticule

Description

Create a graticule. That is, a grid of lon/lat lines that can be used to on a projected map.

The object returned, a `SpatGraticule`, can be plotted with `plot` and `lines`. There is also a `crop` method.

Usage

```
graticule(lon=30, lat=30, crs="")
```

Arguments

lon	numeric. Either a single number (the interval between longitudes), or a vector with longitudes
lat	numeric. Either a single number (the interval between latitudes), or a vector with latitudes
crs	character. The coordinate reference system to use

Value

SpatGraticule

See Also

[plot<SpatGraticule>](#).

Examples

```
g <- graticule(60, 30, crs="+proj=robin")
g

graticule(90, c(-90, -60, -23.5, 0, 23.5, 60, 90), crs="+proj=robin")
```

gridDistance	<i>Distance on a grid</i>
--------------	---------------------------

Description

The function calculates the distance to cells of a SpatRaster when the path has to go through the centers of the eight neighboring raster cells.

The default distance (when scale=1, is meters if the coordinate reference system (CRS) of the SpatRaster is longitude/latitude (+proj=longlat) and in the linear units of the CRS (typically meters) in other cases.

Distances are computed by summing local distances between cells, which are connected with their neighbors in 8 directions.

The shortest distance to the cells with the target value is computed for all cells that are not NA. Cells that are NA cannot be traversed and are ignored, unless the target itself is NA, in which case the distance to the nearest cell that is not NA is computed for all cells that are NA.

Usage

```
## S4 method for signature 'SpatRaster'
gridDist(x, target=0, scale=1, maxiter=50, filename="", ...)
```


Arguments

<code>x</code>	<code>SpatRaster</code>
<code>target</code>	numeric. value of the target cells (where to compute distance to)
<code>scale</code>	numeric. Scale factor. For longitude/latitude data 1 = "m" and 1000 = "km". For planar data that is also the case of the distance unit of the crs is "m"
<code>maxiter</code>	numeric. The maximum number of iterations. Increase this number if you get the warning that <code>costDistance</code> did not converge. Only relevant when target is not NA
<code>filename</code>	character. output filename (optional)
<code>...</code>	additional arguments as for <code>writeRaster</code>

Value

`SpatRaster`

See Also

See [distance](#) for "as the crow flies" distance, and [costDist](#) for distance across a landscape with variable friction

Examples

```
# global lon/lat raster
r <- rast(ncol=10,nrow=10, vals=1)
r[48] <- 0
r[66:68] <- NA
d <- gridDist(r)
plot(d)

# planar
crs(r) <- "+proj=utm +zone=15 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
d <- gridDist(r)
plot(d)

# distance to cells that are not NA
rr <- classify(r, cbind(1, NA))
dd <- gridDist(rr, NA)
```

halo

Add halo-ed text to a plot

Description

Add text to a plot that has a "halo". That is, a buffer around it to enhance visibility.

Usage

```
halo(x, y=NULL, labels, col="black", hc="white", hw=0.1, ...)
```

Arguments

x,y	numeric. coordinates where the text labels should be written
labels	character. The text to be written
col	character. The main color to be used
hc	character. The halo color
hw	numeric. The halo width
...	additional arguments to pass to text

See Also

[text](#), [plot](#)

Examples

```
r <- rast(nrows=4, ncols=4)
values(r) <- 1:ncell(r)
plot(r, col="blue", legend=FALSE)

text(-100, 20, "hello", cex=2)
halo(50, 20, "hello", cex=2)

halo(0, -20, "world", font=3, hc="light blue", cex=2, hw=.2)
halo(0, 90, "world", font=2, cex=2, hw=.2, xpd=TRUE, pos=2)
halo(0, 90, "world", col="white", font=2, hc="blue", cex=2, hw=.2, xpd=TRUE, pos=4)
```

headtail

head and tail of a SpatRaster or SpatVector

Description

Show the head (first values) or tail (last values) of a SpatRaster or of the attributes of a SpatVector.

Usage

```
head(x, ...)
tail(x, ...)
```

Arguments

x	SpatRaster or SpatVector
...	additional arguments passed on to other methods

Value

matrix (SpatRaster) or data.frame (SpatVector)

See Also

[show](#), [geom](#)

Examples

```
r <- rast(nrows=25, ncols=25)
values(r) <- 1:ncell(r)
head(r)
tail(r)
```

hist

Histogram

Description

Create a histogram of the values of a SpatRaster. For large datasets a sample of maxcell is used.

Usage

```
## S4 method for signature 'SpatRaster'
hist(x, layer, maxcell=1000000, plot=TRUE, maxnl=16, main, ...)
```

Arguments

x	SpatRaster
layer	positive integer or character to indicate layer numbers (or names). If missing, all layers up to maxnl are used
maxcell	integer. To regularly sample very large objects
plot	logical. Plot the histogram or only return the histogram values
maxnl	positive integer. The maximum number of layers to use. Ignored if layer is not missing
main	character. Main title(s) for the plot. Default is the value of names
...	additional arguments. See hist

Value

This function is principally used for plotting a histogram, but it also returns an object of class "histogram" (invisibly if plot=TRUE).

See Also

[pairs](#), [boxplot](#)

Examples

```

r1 <- r2 <- rast(nrows=50, ncols=50)
values(r1) <- runif(ncell(r1))
values(r2) <- runif(ncell(r1))
rs <- r1 + r2
rp <- r1 * r2

opar <- par(no.readonly =TRUE)
par(mfrow=c(2,2))
plot(rs, main='sum')
plot(rp, main='product')
hist(rs)
a <- hist(rp)
a
x <- c(rs, rp, sqrt(rs))
hist(x)
par(opar)

```

ifel

ifelse for SpatRasters

Description

Implementation of [ifelse](#) for SpatRasters. This method allows for a concise expression of what can otherwise be achieved with a combination of [classify](#), [mask](#), and [cover](#).

ifel is an R equivalent to the Con method in ArcGIS (arcpy).

Usage

```

## S4 method for signature 'SpatRaster'
ifel(test, yes, no, filename="", ...)

```

Arguments

test	SpatRaster
yes	SpatRaster or numeric
no	SpatRaster or numeric
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Examples

```

r <- rast(nrows=5, ncols=5, xmin=0, xmax=1, ymin=0, ymax=1)
values(r) <- c(-10:0, NA, NA, NA, 0:10)

x <- ifel(r > 1, 1, r)
# same as
a <- classify(r, cbind(1, Inf, 1))
# or
b <- app(r, fun=function(i) {i[i > 1] <- 1; i})
# or
d <- clamp(r, -Inf, 1)
# or (not recommended for large datasets)
e <- r
e[e>1] <- 1

## other examples
f <- ifel(is.na(r), 100, r)

z <- ifel(r > -2 & r < 2, 100, 0)

# nested expressions
y <- ifel(r > 1, 1, ifel(r < -1, -1, r))

k <- ifel(r > 0, r+10, ifel(r < 0, r-10, 3))

```

image

*SpatRaster image method***Description**

Plot (make a map of) the values of a *SpatRaster* via [image](#). See [plot](#) if you need more fancy options such as a legend.

Usage

```

## S4 method for signature 'SpatRaster'
image(x, y=1, maxcell=500000, ...)

```

Arguments

x	<i>SpatRaster</i>
y	positive integer indicating the layer to be plotted, or a character indicating the name of the layer
maxcell	positive integer. Maximum number of cells to use for the plot
...	additional arguments as for graphics:: image

See Also

[plot](#)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
image(r)
image(r, col=rainbow(24))
```

impose	<i>Impose the geometry of a SpatRaster to those in a SpatRasterCollection.</i>
--------	--

Description

Warp the members of a SpatRasterCollection to match the geometry of a SpatRaster.

Usage

```
## S4 method for signature 'SpatRasterCollection'
impose(x, y, filename="", ...)
```

Arguments

x	SpatRasterCollection
y	SpatRaster
filename	character. Output filename
...	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

[resample](#)

initialize	<i>Initialize a SpatRaster with values</i>
------------	--

Description

Create a SpatRaster with values reflecting a cell property: 'x', 'y', 'col', 'row', 'cell' or 'chess'. Alternatively, a function can be used. In that case, cell values are initialized without reference to pre-existing values. E.g., initialize with a random number (fun=[runif](#)). While there are more direct ways of achieving this for small objects (see examples) for which a vector with all values can be created in memory, the init function will also work for SpatRaster objects with many cells.

Usage

```
## S4 method for signature 'SpatRaster'
init(x, fun, filename="", ...)
```

Arguments

x	SpatRaster
fun	function to be applied. This must be a either single number, multiple numbers, a function, or one of a set of known character values. A function must take the number of cells as a single argument to return a vector of values with a length equal to the number of cells, such as fun= runif . Allowed character values are 'x', 'y', 'row', 'col', 'cell', and 'chess' to get the x or y coordinate, row, col or cell number or a chessboard pattern (alternating 0 and 1 values)
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Examples

```
r <- rast(ncols=10, nrows=5, xmin=0, xmax=10, ymin=0, ymax=5)
x <- init(r, fun="cell")
y <- init(r, fun=runif)

# initialize with a single value
z <- init(r, fun=8)
```

inplace

*Change values in-place***Description**

These "in-place" replacement methods assign new value to an object without making a copy. That is efficient, but if there is a copy of the object that you made by standard assignment (e.g. with `y <- x`), that copy is also changed.

`set.names` is the in-place replacement version of `names<-`.

`set.ext` is the in-place replacement version of `ext<-`

`set.values` is the in-place replacement version of `[<-`.

`set.cats` is the in-place replacement version of `categories`

`set.crs` is the in-place replacement version of `crs<-`

Usage

```
## S4 method for signature 'SpatRaster'
set.names(x, value, index=1:nlyr(x), validate=FALSE)
## S4 method for signature 'SpatRasterDataset'
set.names(x, value, index=1:length(x), validate=FALSE)
## S4 method for signature 'SpatVector'
set.names(x, value, index=1:ncol(x), validate=FALSE)

## S4 method for signature 'SpatRaster'
set.ext(x, value)
## S4 method for signature 'SpatVector'
set.ext(x, value)

## S4 method for signature 'SpatRaster'
set.crs(x, value)
## S4 method for signature 'SpatVector'
set.crs(x, value)

## S4 method for signature 'SpatRaster'
set.values(x, cells, values, layer=0)

## S4 method for signature 'SpatRaster'
set.cats(x, layer=1, value, active=1)

## S4 method for signature 'SpatRaster'
set.RGB(x, value, type="rgb")
```

Arguments

`x` `SpatRaster`

value	character for set.names. For set.cats: a data.frame with columns (value, category) or vector with category names. For set.RGB 3 or 4 numbers indicating the RGB(A) layers
index	positive integer indicating layer(s) to assign a name to
validate	logical. Make names valid and/or unique?
cells	cell numbers or missing
values	replacement values or missing to load all values into memory
layer	positive integer(s) indicating to which layer(s) to you want to assign these categories or to which you want to set these values. A number < 1 indicates "all layers"
active	positive integer indicating the active category (column number in value, but not counting the first column)
type	character. The color space. One of "rgb" "hsv", "hsi" and "hsl"

Value

logical (invisibly)

Examples

```
s <- rast(ncols=5, nrows=5, nlyrs=3)
x <- s
names(s)
names(s) <- c("a", "b", "c")
names(s)
names(x)

x <- s
set.names(s, c("e", "f", "g"))
names(s)
names(x)

set.ext(x, c(0,180,0,90))

f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)

#values from file to memory
set.values(r)

# change values
set.values(r, 1:1000, 900)
```

inset

*Make an inset map***Description**

Make an inset map or scale the extent of a SpatVector

Usage

```
## S4 method for signature 'SpatVector'
inset(x, e, loc="", scale=0.2, background="white",
      perimeter=TRUE, box=NULL, pper, pbox, ...)
```

```
## S4 method for signature 'SpatRaster'
inset(x, e, loc="", scale=0.2, background="white",
      perimeter=TRUE, box=NULL, pper, pbox, ...)
```

```
## S4 method for signature 'SpatVector'
inext(x, e, y=NULL, gap=0)
```

Arguments

x	SpatVector, SpatRaster
e	SpatExtent to set the size and location of the inset. Or missing
loc	character. One of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center"
scale	numeric. The relative size of the inset, used when x is missing
background	color for the background of the inset. Use NA for no background color
perimeter	logical. If TRUE a perimeter (border) is drawn around the inset
box	SpatExtent or missing, to draw a box on the inset, e.g. to show where the map is located in a larger area
pper	list with graphical parameters (arguments) such as col and lwd for the perimeter line
pbox	list with graphical parameters (arguments) such as col and lwd for the box (line)
...	additional arguments passed to plot for the drawing of x
y	SpatVector. If not NULL, y is scaled based with the parameters for x. This is useful, for example, when x represent boundaries, and y points within these boundaries
gap	numeric to add space between the SpatVector and the SpatExtent

Value

scaled and shifted SpatVector or SpatRaster (returned invisibly)

See Also[sbar, rescale, shift](#)**Examples**

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
x <- v[v$NAME_2 == "Diekirch", ]

plot(x, density=10, col="blue")
inset(v)

# more elaborate
plot(x, density=10, col="blue")
inset(v, col = "brown", border="lightgrey", perimeter=TRUE,
      pper=list(col="orange", lwd=3, lty=2),
      box=ext(x), pbox=list(col="blue", lwd=2))

cols <- rep("light grey", 12)
cols[2] <- "red"
e <- ext(c(6.2, 6.3, 49.9, 50))
b <- ext(x)+0.02
inset(v, e=e, col=cols, box=b)

# with a SpatRaster
ff <- system.file("ex/elev.tif", package="terra")
r <- rast(ff)
r <- crop(r, ext(x) + .01)
plot(r, type="int", mar=c(2,2,2,2), plg=list(x="topright"))
lines(v, lwd=1.5)
lines(x, lwd=2.5)
inset(v, col=cols, loc="topleft", scale=0.15)

# a more complex one
plot(r, plg=list(title="meter\n", shrink=.2, cex=.8))
lines(v, lwd=4, col="white")
lines(v, lwd=1.5)
lines(x, lwd=2.5)
text(x, "NAME_2", cex=1.5, halo=TRUE)
sbar(6, c(6.04, 49.785), type="bar", below="km", label=c(0,3,6), cex=.8)
s <- inset(v, col=cols, box=b, scale=.2, loc="topright", background="light yellow",
          pbox=list(lwd=2, lty=5, col="blue"))

# note the returned inset SpatVector
s
lines(s, col="orange")
```

Description

Interpolate points within a moving window using inverse distance weighting. The maximum number of points used can be restricted, optionally by selecting the nearest points.

Usage

```
## S4 method for signature 'SpatRaster,SpatVector'
interpIDW(x, y, field, radius, power=2, smooth=0,
          maxPoints=Inf, minPoints=1, near=FALSE, fill=NA, filename="", ...)

## S4 method for signature 'SpatRaster,matrix'
interpIDW(x, y, radius, power=2, smooth=0,
          maxPoints=Inf, minPoints=1, near=FALSE, fill=NA, filename="", ...)
```

Arguments

x	SpatRaster
y	SpatVector or matrix with three columns (x,y,z)
field	character. field name in SpatVector y
radius	numeric. The radius of the circle (single number). If near=FALSE, it is also possible to use two or three numbers. Two numbers are interpreted as the radii of an ellipse (x and y-axis). A third number should indicated the desired, counter clockwise, rotation of the ellipse (in degrees)
power	numeric. Weighting power
smooth	numeric. Smoothing parameter
minPoints	numeric. The minimum number of points to use. If fewer points are found in a search ellipse it is considered empty and the fill value is returned
maxPoints	numeric. The maximum number of points to consider in a search area. Additional points are ignored. If fewer points are found, the fill value is returned
near	logical. If maxPoints is reached, should the nearest points within the neighborhood be used?
fill	numeric. value to use to fill empty cells
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[rasterizeWin](#), [rasterize](#), [interpNear](#), [interpolate](#)

Examples

```

r <- rast(ncol=100, nrow=100, crs="local", xmin=0, xmax=50, ymin=0, ymax=50)
set.seed(100)
x <- runif(25, 5, 45)
y <- runif(25, 5, 45)
z <- sample(25)
xyz <- cbind(x,y,z)

x <- interpIDW(r, xyz, radius=5, power=1, smooth=1, maxPoints=5)

```

interpNear	<i>Nearest neighbor interpolation</i>
------------	---------------------------------------

Description

Nearest neighbor interpolation of points, using a moving window

Usage

```

## S4 method for signature 'SpatRaster,SpatVector'
interpNear(x, y, field, radius, interpolate=FALSE, fill=NA, filename="", ...)

## S4 method for signature 'SpatRaster,matrix'
interpNear(x, y, radius, interpolate=FALSE, fill=NA, filename="", ...)

```

Arguments

x	SpatRaster
y	SpatVector or matrix with three columns (x,y,z)
field	character. field name in SpatVector y
radius	numeric. The radius of the circle (single number). If interpolate=FALSE it is also possible to use two or three numbers. Two numbers are interpreted as the radii of an ellipse (x and y-axis). A third number should indicated the desired, counter clockwise, rotation of the ellipse (in degrees)
interpolate	logical. Should the nearest neighbor values be linearly interpolated between points?
fill	numeric. value to use to fill empty cells
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[rasterizeWin](#), [rasterize](#), [interpIDW](#), [interpolate](#)

Examples

```
r <- rast(ncol=100, nrow=100, crs="local", xmin=0, xmax=50, ymin=0, ymax=50)
set.seed(100)
x <- runif(25, 5, 45)
y <- runif(25, 5, 45)
z <- sample(25)
xyz <- cbind(x,y,z)

x <- interpNear(r, xyz, radius=5)

p <- vect(data.frame(xyz), geom=c("x", "y"))
v <- voronoi(p)

plot(x, col=rainbow(25))
lines(v)

# plot(v, col=rainbow(25)); points(p)
```

interpolation

Spatial interpolation

Description

Make a `SpatRaster` with interpolated values using a fitted model object of classes such as "gstat" (gstat package) or "Krig" (fields package), or any other model that has location (e.g., "x" and "y", or "longitude" and "latitude") as predictors (independent variables). If x and y are the only predictors, it is most efficient if you provide an empty (no associated data in memory or on file) `SpatRaster` for which you want predictions. If there are more spatial predictor variables provide these as a `SpatRaster` in the first argument of the function. If you do not have x and y locations as implicit predictors in your model you should use [predict](#) instead.

Usage

```
## S4 method for signature 'SpatRaster'
interpolate(object, model, fun=predict, ..., xyNames=c("x", "y"),
            factors=NULL, const=NULL, index = NULL, cores=1, cpkgs=NULL,
            na.rm=FALSE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

object	<code>SpatRaster</code>
model	model object

<code>fun</code>	function. Default value is "predict", but can be replaced with e.g. "predict.se" (depending on the class of <code>model</code>), or a custom function (see examples)
<code>...</code>	additional arguments passed to <code>fun</code>
<code>xyNames</code>	character. variable names that the model uses for the spatial coordinates. E.g., <code>c("longitude", "latitude")</code>
<code>factors</code>	list with levels for factor variables. The list elements should be named with names that correspond to names in <code>object</code> such that they can be matched. This argument may be omitted for some models from which the levels can be extracted from the model object
<code>const</code>	data.frame. Can be used to add a constant for which there is no <code>SpatRaster</code> for model predictions. This is particularly useful if the constant is a character-like factor value
<code>index</code>	positive integer or <code>NULL</code> . Allows for selecting of the variable returned if the model returns multiple variables
<code>cores</code>	positive integer. If <code>cores > 1</code> , a 'parallel' package cluster with that many cores is created and used
<code>cpkgs</code>	character. The package(s) that need to be loaded on the nodes to be able to run the <code>model.predict</code> function (see examples in predict)
<code>na.rm</code>	logical. If <code>TRUE</code> , cells with NA values in the predictors are removed from the computation. This option prevents errors with models that cannot handle NA values. In most other cases this will not affect the output. An exception is when predicting with a model that returns predicted values even if some (or all!) variables are NA
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If <code>TRUE</code> , filename is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster

Value

`SpatRaster`

See Also

[predict](#), `link{interpIDW}`, `link{interpNear}`

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
ra <- aggregate(r, 10)
xy <- data.frame(xyFromCell(ra, 1:ncell(ra)))
v <- values(ra)
i <- !is.na(v)
xy <- xy[i,]
v <- v[i]

## Not run:
```

```

library(fields)
tps <- Tps(xy, v)
p <- rast(r)

# use model to predict values at all locations
p <- interpolate(p, tps)
p <- mask(p, r)
plot(p)

### change "fun" from predict to fields::predictSE to get the TPS standard error
## need to use "rast(p)" to remove the values
se <- interpolate(rast(p), tps, fun=predictSE)
se <- mask(se, r)
plot(se)

### another predictor variable, "e"
e <- (init(r, "x") * init(r, "y")) / 100000000
names(e) <- "e"

z <- as.matrix(extract(e, xy)[,-1])

## add as another independent variable
xyz <- cbind(xy, z)
tps2 <- Tps(xyz, v)
p2 <- interpolate(e, tps2, xyOnly=FALSE)

## as a linear coveriate
tps3 <- Tps(xy, v, Z=z)

## Z is a separate argument in Krig.predict, so we need a new function
## Internally (in interpolate) a matrix is formed of x, y, and elev (Z)

pfun <- function(model, x, ...) {
  predict(model, x[,1:2], Z=x[,3], ...)
}
p3 <- interpolate(e, tps3, fun=pfun)

#### gstat examples
library(gstat)
library(sp)
data(meuse)

### inverse distance weighted (IDW)
r <- rast(system.file("ex/meuse.tif", package="terra"))
mg <- gstat(id = "zinc", formula = zinc~1, locations = ~x+y, data=meuse,
           nmax=7, set=list(idp = .5))
z <- interpolate(r, mg, debug.level=0, index=1)
z <- mask(z, r)

## with a model built with an `sf` object you need to provide custom function

library(sf)

```



```

sfmeuse <- st_as_sf(meuse, coords = c("x", "y"), crs=crs(r))
mgf <- gstat(id = "zinc", formula = zinc~1, data=sfmeuse, nmax=7, set=list(idp = .5))

interpolate_gstat <- function(model, x, crs, ...) {
  v <- st_as_sf(x, coords=c("x", "y"), crs=crs)
  p <- predict(model, v, ...)
  as.data.frame(p)[1:2]
}

zsf <- interpolate(r, mgf, debug.level=0, fun=interpolate_gstat, crs=crs(r), index=1)
zsf <- mask(zsf, r)

### kriging

### ordinary kriging
v <- variogram(log(zinc)~1, ~x+y, data=meuse)
mv <- fit.variogram(v, vgm(1, "Sph", 300, 1))
gOK <- gstat(NULL, "log.zinc", log(zinc)~1, meuse, locations=~x+y, model=mv)
OK <- interpolate(r, gOK, debug.level=0)

## universal kriging
vu <- variogram(log(zinc)~elev, ~x+y, data=meuse)
mu <- fit.variogram(vu, vgm(1, "Sph", 300, 1))
gUK <- gstat(NULL, "log.zinc", log(zinc)~elev, meuse, locations=~x+y, model=mu)
names(r) <- "elev"
UK <- interpolate(r, gUK, debug.level=0)

## co-kriging
gCoK <- gstat(NULL, 'log.zinc', log(zinc)~1, meuse, locations=~x+y)
gCoK <- gstat(gCoK, 'elev', elev~1, meuse, locations=~x+y)
gCoK <- gstat(gCoK, 'cadmium', cadmium~1, meuse, locations=~x+y)
gCoK <- gstat(gCoK, 'copper', copper~1, meuse, locations=~x+y)
coV <- variogram(gCoK)
plot(coV, type='b', main='Co-variogram')
coV.fit <- fit.lmc(coV, gCoK, vgm(model='Sph', range=1000))
coV.fit
plot(coV, coV.fit, main='Fitted Co-variogram')
coK <- interpolate(r, coV.fit, debug.level=0)
plot(coK)

## End(Not run)

```

intersect

Intersection

Description

You can intersect `SpatVectors` with each other or with a `SpatExtent`. Intersecting points with points uses the extent of `y` to get the intersection. Intersecting of points and lines is not supported because of numerical inaccuracies with that. You can use [buffer](#), to create polygons from lines and use these with `intersect`.

You can also intersect two `SpatExtents`.

When intersecting two `SpatRasters` these need to be aligned (have the same origin and spatial resolution). The values of the returned `SpatRaster` are `TRUE` where both have input rasters have values, `FALSE` where one has values, and `NA` in all other cells.

See `crop` for intersection of a `SpatRaster` with a `SpatExtent` (or the extent of a `SpatRaster` or `SpatVector`).

See `is.related(x, y, "intersects")` to find out which geometries of a `SpatVector` intersect. You can spatially subset a `SpatVector` with another one with `x[y]`

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
intersect(x, y)

## S4 method for signature 'SpatVector,SpatExtent'
intersect(x, y)

## S4 method for signature 'SpatExtent,SpatVector'
intersect(x, y)

## S4 method for signature 'SpatExtent,SpatExtent'
intersect(x, y)

## S4 method for signature 'SpatRaster,SpatRaster'
intersect(x, y)
```

Arguments

<code>x</code>	<code>SpatVector</code> , <code>SpatExtent</code> , or <code>SpatRaster</code>
<code>y</code>	<code>SpatVector</code> , <code>SpatExtent</code> , or <code>SpatRaster</code> if <code>x</code> is a <code>SpatRaster</code>

Value

Same as `x`

See Also

`union`, `crop`, `relate`, `[`

Examples

```
e1 <- ext(-10, 10, -20, 20)
e2 <- ext(0, 20, -40, 5)
intersect(e1, e2)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
e <- ext(5.6, 6, 49.55, 49.7)
x <- intersect(v, e)
```

```

p <- vect(c("POLYGON ((5.8 49.8, 6 49.9, 6.15 49.8, 6 49.6, 5.8 49.8))",
"POLYGON ((6.3 49.9, 6.2 49.7, 6.3 49.6, 6.5 49.8, 6.3 49.9))"), crs=crs(v))
values(p) <- data.frame(pid=1:2, area=expanses(p))

y <- intersect(v, p)

r <- s <- rast(ncol=5, nrow=5, xmin=1, xmax=5, ymin=1, ymax=5)
r[5:20] <- 5:20
s[11:20] <- 11:20
rs <- intersect(r, s)

u <- shift(r, .8)
us <- intersect(u, s)

```

is.bool

*Raster value types***Description**

The values in a `SpatRaster` layer are by default numeric, but they can also be logical (Boolean), integer, or categorical (factor).

For a `SpatRaster`, `as.logical` and `isTRUE` is equivalent to `as.bool`. `isFALSE` is equivalent to `!as.bool`, and `as.integer` is the same as `as.int`.

See [levels](#) and [cats](#) to create categorical layers by setting labels.

Usage

```

## S4 method for signature 'SpatRaster'
is.bool(x)

## S4 method for signature 'SpatRaster'
as.bool(x, filename, ...)

## S4 method for signature 'SpatRaster'
is.int(x)

## S4 method for signature 'SpatRaster'
as.int(x, filename, ...)

## S4 method for signature 'SpatRaster'
is.factor(x)

## S4 method for signature 'SpatRaster'
as.factor(x)

```

Arguments

<code>x</code>	SpatRaster
<code>filename</code>	character. Output filename
<code>...</code>	list with named options for writing files as in writeRaster

Value

The `as.*` methods return a new SpatRaster, whereas the `is.*` methods return a logical value for each layer in `x`.

See Also

[levels](#) and [cats](#) to create categorical layers (and set labels).

Examples

```
r <- rast(nrows=10, ncols=10, vals=1:100)
is.bool(r)
z <- as.bool(r)
is.bool(z)

x <- r > 25
is.bool(x)

rr <- r/2
is.int(rr)
is.int(round(rr))
```

is.lonlat

Check for longitude/latitude crs

Description

Test whether a SpatRaster or SpatVector has a longitude/latitude coordinate reference system (CRS), or perhaps has one. That is, when the CRS is unknown ("") but the x coordinates are within -181 and 181 and the y coordinates are within -90.1 and 90.1. For a SpatRaster you can also test if it has a longitude/latitude CRS and it is "global" (covers all longitudes).

A warning is given if the CRS is missing or if it is specified as longitude/latitude but the coordinates do not match that.

Usage

```
## S4 method for signature 'SpatRaster'
is.lonlat(x, perhaps=FALSE, warn=TRUE, global=FALSE)

## S4 method for signature 'SpatVector'
is.lonlat(x, perhaps=FALSE, warn=TRUE)
```

```
## S4 method for signature 'character'
is.lonlat(x, perhaps=FALSE, warn=TRUE)
```

Arguments

x	SpatRaster or SpatVector
perhaps	logical. If TRUE and the CRS is unknown, the method returns TRUE if the coordinates are plausible for longitude/latitude
warn	logical. If TRUE, a warning is given if the CRS is unknown but assumed to be lon/lat and perhaps=TRUE
global	logical. If TRUE, the method tests if the raster covers all longitudes (from -180 to 180 degrees) such that the extreme columns are in fact adjacent

Value

logical or NA

Examples

```
r <- rast()
is.lonlat(r)
is.lonlat(r, global=TRUE)

crs(r) <- ""
is.lonlat(r)
is.lonlat(r, perhaps=TRUE, warn=FALSE)

crs(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
is.lonlat(r)
```

lapp	<i>Apply a function to layers of a SpatRaster, or sub-datasets of a SpatRasterDataset</i>
------	---

Description

Apply a function to a SpatRaster, using layers as arguments.

The number of arguments in function fun must match the number of layers in the SpatRaster (or the number of sub-datasets in the SpatRasterDataset). For example, if you want to multiply two layers, you could use this function: fun=function(x,y){return(x*y)} percentage: fun=function(x,y){return(100 * x / y)}. If you combine three layers you could use fun=function(x,y,z){return((x + y) * z)}

Before you use the function, test it to make sure that it is vectorized. That is, it should work for vectors longer than one, not only for single numbers. Or if the input SpatRaster(s) have multiple layers, it should work for a matrix (multiple cells) of input data (or matrices in the case of a SpatRasterDataSet). The function must return the same number of elements as its input vectors, or

multiples of that. Also make sure that the function is NA-proof: it should return the same number of values when some or all input values are NA. And the function must return a vector or a matrix, not a data.frame. To test it, run it with `do.call(fun, data)` (see examples).

Use [app](#) for summarize functions such as `sum`, that take any number of arguments; and [tapp](#) to do so for groups of layers.

Usage

```
## S4 method for signature 'SpatRaster'
lapp(x, fun, ..., usenames=FALSE, cores=1, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterDataset'
lapp(x, fun, ..., usenames=FALSE, recycle=FALSE,
      filename="", overwrite=FALSE, wopt=list())
```

Arguments

<code>x</code>	SpatRaster or SpatRasterDataset
<code>fun</code>	a function that takes a vector and can be applied to each cell of <code>x</code>
<code>...</code>	additional arguments to be passed to <code>fun</code>
<code>usenames</code>	logical. Use the layer names (or dataset names if <code>x</code> is a SpatRasterDataset) to match the function arguments? If FALSE, argument matching is by position
<code>cores</code>	positive integer. If <code>cores > 1</code> , a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object
<code>recycle</code>	logical. Recycle layers to match the subdataset with the largest number of layers
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster

Value

SpatRaster

Note

Use [sapp](#) or `lapply` to apply a function that takes a SpatRaster as argument to each layer of a SpatRaster (that is rarely necessary).

See Also

[app](#), [tapp](#), [math](#)

Examples

```

s <- rast(system.file("ex/logo.tif", package="terra")) + 1
ss <- s[[2:1]]

fvi <- function(x, y){ (x - y) / (x + y) }
# test the function
data <- list(c(1:5,NA), 6:1)
do.call(fvi, data)

x <- lapp(ss, fun=fvi )

# which is the same as supplying the layers to "fun"
# in some cases this will be much faster
y <- fvi(s[[2]], s[[1]])

f2 <- function(x, y, z){ (z - y + 1) / (x + y + 1) }
p1 <- lapp(s, fun=f2 )

p2 <- lapp(s[[1:2]], f2, z=200)

# the usenames argument

fvi2 <- function(red, green){ (red - green) / (red + green) }
names(s)
x1 <- lapp(s[[1:2]], fvi2, usenames=TRUE)
x2 <- lapp(s[[2:1]], fvi2, usenames=TRUE)
# x1 and x2 are the same, despite the change in the order of the layers
# x4 is also the same, but x3 is not
x3 <- lapp(s[[2:1]], fvi2, usenames=FALSE)
x4 <- lapp(s, fvi2, usenames=TRUE)

# while this would fail because
# there are too many layers in s
# x5 <- lapp(s, fvi2, usenames=FALSE)

pairs(c(x1, x2, x3, x4))

## SpatRasterDataset
x <- sds(s, s[[1]]+50)
fun <- function(x, y) { x/y }

# test "fun"
data <- list(matrix(1:9, ncol=3), matrix(9:1, ncol=3))
do.call(fun, data)

lapp(x, fun, recycle=TRUE)

# the same, more concisely
z <- s / (s[[1]]+50)

```

layerCor

*Correlation and (weighted) covariance***Description**

Compute correlation, (weighted) covariance, or similar summary statistics that compare the values of all pairs of the layers of a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
layerCor(x, fun, w, asSample=TRUE, na.rm=FALSE, maxcell=Inf, ...)
```

Arguments

x	SpatRaster
fun	character. The statistic to compute: either "cov" (covariance), "weighted.cov" (weighted covariance), or "pearson" (correlation coefficient) or your own function that takes two vectors as argument to compute a single number
w	SpatRaster with the weights to compute the weighted covariance. It should have a single layer and the same geometry as x
asSample	logical. If TRUE, the statistic for a sample (denominator is n-1) is computed, rather than for the population (denominator is n). Only for the standard functions
na.rm	logical. Should missing values be removed?
maxcell	postive integer. The number of cells to be regularly sampled. Only used when fun is a function
...	additional arguments for fun (if it is a proper function)

Value

If fun is one of the three standard statistics, you get a list with two items: the correlation or (weighted) covariance matrix, and the (weighted) means.

If fun is a function, you get a matrix.

References

For the weighted covariance:

- Canty, M.J. and A.A. Nielsen, 2008. Automatic radiometric normalization of multitemporal satellite imagery with the iteratively re-weighted MAD transformation. Remote Sensing of Environment 112:1025-1036.
- Nielsen, A.A., 2007. The regularized iteratively reweighted MAD method for change detection in multi- and hyperspectral data. IEEE Transactions on Image Processing 16(2):463-478.

See Also[global](#), [cov.wt](#), [weighted.mean](#)**Examples**

```
b <- rast(system.file("ex/logo.tif", package="terra"))
layerCor(b, "pearson")

layerCor(b, "cov")

# weigh by column number
w <- init(b, fun="col")
layerCor(b, "weighted.cov", w=w)
```

linearUnits*Linear units of the coordinate reference system*

Description

Get the linear units of the coordinate reference system (crs) of a `SpatRaster` or `SpatVector` expressed in m. The value returned is used internally to transform area and perimeter measures to meters. The value returned for longitude/latitude crs is zero.

Usage

```
## S4 method for signature 'SpatRaster'
linearUnits(x)

## S4 method for signature 'SpatVector'
linearUnits(x)
```

Arguments

x `SpatRaster` or `SpatVector`

Value

numeric (meter)

See Also[crs](#)

Examples

```
x <- rast()
crs(x) <- ""
linearUnits(x)

crs(x) <- "+proj=longlat +datum=WGS84"
linearUnits(x)

crs(x) <- "+proj=utm +zone=1 +units=cm"
linearUnits(x)

crs(x) <- "+proj=utm +zone=1 +units=km"
linearUnits(x)

crs(x) <- "+proj=utm +zone=1 +units=us-ft"
linearUnits(x)
```

lines

Add points, lines, or polygons to a map

Description

Add a vector geometries to a plot (map) with points, lines, or polys.

These are simpler alternatives for `plot(x, add=TRUE)`

These methods also work for a small(!) `SpatRaster`. Only cells that are not NA in the first layer are used.

Usage

```
## S4 method for signature 'SpatVector'
points(x, col, cex=0.7, pch=16, alpha=1, ...)

## S4 method for signature 'SpatVector'
lines(x, y=NULL, col, lwd=1, lty=1, arrows=FALSE, alpha=1, ...)

## S4 method for signature 'SpatVector'
polys(x, col, border="black", lwd=1, lty=1, alpha=1, ...)

## S4 method for signature 'SpatRaster'
points(x, ...)

## S4 method for signature 'SpatRaster'
lines(x, mx=10000, ...)

## S4 method for signature 'SpatRaster'
polys(x, mx=10000, dissolve=TRUE, ...)
```

```
## S4 method for signature 'SpatExtent'
points(x, col="black", alpha=1, ...)

## S4 method for signature 'SpatExtent'
lines(x, col="black", alpha=1, ...)

## S4 method for signature 'SpatExtent'
polys(x, col, alpha=1, ...)
```

Arguments

x	SpatVector or SpatExtent
y	missing or SpatVector. If both x and y have point geometry and the same number of rows, lines are drawn between pairs of points
col	character. Colors
border	character. color(s) of the polygon borders. Use NULL or NA to not draw a border
cex	numeric. point size magnifier. See par
pch	positive integer, point type. See points . On some (linux) devices, the default symbol "16" is not a very smooth circle. You can use "20" instead (it takes a bit longer to draw) or "1" for an open circle
alpha	number between 0 and 1 to set transparency
lwd	numeric, line-width. See par
lty	positive integer, line type. See par
arrows	logical. If TRUE and y is a SpatVector, arrows are drawn instead of lines. See ?arrows for additional arguments
mx	positive number. If the number of cells of SpatRaster x is higher, the method will fail with an error message
dissolve	logical. Should boundaries between cells with the same value be removed?
...	additional graphical arguments such as lwd, cex and pch

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

r <- rast(v)
values(r) <- 1:ncell(r)
plot(r)
lines(v)
points(v)
```

makeTiles

*Make tiles***Description**

Divide a `SpatRaster` into "tiles". The cells of another `SpatRaster` (normally with a much lower resolution) or a `SpatVector` with polygon geometry can be used to define the tiles. You can also provide one or two numbers to indicate the number of rows and columns per tile.

Usage

```
## S4 method for signature 'SpatRaster'
makeTiles(x, y, filename="tile_.tif", extend=FALSE, na.rm=FALSE, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>y</code>	<code>SpatRaster</code> , <code>SpatVector</code> , or numeric
<code>filename</code>	character. Output filename template. Filenames will be altered by adding the tile number for each tile
<code>extend</code>	logical. If TRUE, the extent of <code>y</code> is expanded to assure that it covers all of <code>x</code>
<code>na.rm</code>	logical. If TRUE, tiles with only missing values are ignored
<code>...</code>	additional arguments for writing files as in writeRaster

Value

character (filenames)

See Also

[vrt](#) to create a virtual raster from tiles

Examples

```
r <- rast(ncols=100, nrows=100)
values(r) <- 1:ncell(r)
x <- rast(ncols=2, nrows=2)
filename <- paste0(tempfile(), "__.tif")
ff <- makeTiles(r, x, filename)
ff

vrt(ff)
```

makeVRT	<i>Make a VRT header file</i>
---------	-------------------------------

Description

Create a VRT header file for a "flat binary" raster file that needs a header file to be able to read it, but does not have it.

Usage

```
makeVRT(filename, nrow, ncol, nlyr=1, extent, xmin, ymin, xres, yres=xres, xycenter=TRUE,
        crs="+proj=longlat", lyrnms="", datatype, NAflag=NA, bandorder="BIL", byteorder="LSB",
        toptobottom=TRUE, offset=0, scale=1)
```

Arguments

filename	character. raster filename (without the ".vrt" extension)
nrow	positive integer, the number of rows
ncol	positive integer, the number of columns
nlyr	positive integer, the number of layers
extent	SpatExtent or missing
xmin	numeric. minimum x coordinate (only used if extent is missing)
ymin	numeric. minimum y coordinate (only used if extent is missing)
xres	positive number. x resolution
yres	positive number. y resolution)
xycenter	logical. If TRUE, xmin and xmax represent the coordinates of the center of the extreme cell, in stead of the coordinates of the outside corner. Only used of extent is missing
crs	character. Coordinate reference system description
lyrnms	character. Layer names
datatype	character. One of "INT2S", "INT4S", "INT1U", "INT2U", "INT4U", "FLT4S", "FLT8S". If missing, this is guessed from the file size (INT1U for 1 byte per value, INT2S for 2 bytes and FLT4S for 4 bytes per value). This may be wrong because, for example, 2 bytes per value may in fact be INT2U (with the U for unsigned) values
NAflag	numeric. The value used as the "NA flag"
bandorder	character. One of "BIL", "BIP", or "BSQ". That is Band Interleaved by Line, or by Pixel, or Band SeQuential
byteorder	character. One of "LSB", "MSB". "MSB" is common for files generated on Linux systems, whereas "LSB" is common for files generated on windows
toptobottom	logical. If FALSE, the values are read bottom to top
offset	numeric. offset to be applied
scale	numeric. scale to be applied

Value

character (.VRT filename)

See Also

[vrt](#) to create a vrt for a collection of raster tiles

map.pal	<i>map.pal arrow</i>
---------	----------------------

Description

Get a color palette for mapping. These palettes were copied from GRASS.

Usage

map.pal(name, n=50, ...)

Arguments

name	character (name of a palette, see Details), or missing (to get the available names)
n	numeric. The number of colors
...	additional arguments that are passed to colorRamp

Details

Name	Description
aspect	aspect oriented grey colors
bcyr	blue through cyan through yellow to red
bgyr	blue through green through yellow to red
blues	white to blue
byg	blue through yellow to green
byr	blue through yellow to red
curvature	for terrain curvatures
differences	differences oriented colors
elevation	maps relative ranges of raster values to elevation color ramp
grass	GRASS GIS green (perceptually uniform)
greens	white to green
grey	grey scale
grey.eq	histogram-equalized grey scale
grey.log	histogram logarithmic transformed grey scale
gyr	green through yellow to red
haxby	relative colors for bathymetry or topography
inferno	perceptually uniform sequential color table inferno
magma	perceptually uniform sequential color table magma

oranges	white to orange
plasma	perceptually uniform sequential color table plasma
rainbow	rainbow color table
ramp	color ramp
random	random color table
reds	white to red
roygbiv	
rstcurv	terrain curvature
ryb	red through yellow to blue
ryg	red through yellow to green
sepia	yellowish-brown through to white
viridis	perceptually uniform sequential color table viridis
water	water depth
wave	color wave

Value

none

See Also

[terrain.colors](#)

Examples

```
map.pal("elevation", 10)

r <- rast(system.file("ex/elev.tif", package="terra"))
plot(r, col=map.pal("elevation"))

map.pal()
```

mask	<i>Mask values in a SpatRaster or SpatVector</i>
------	--

Description

If x is a SpatRaster: Create a new SpatRaster that has the same values as SpatRaster x, except for the cells that are NA (or other maskvalue) in another SpatRaster (the 'mask'), or the cells that are not covered by a SpatVector. These cells become NA (or another updatevalue).

If x is a SpatVector: Select geometries of x that intersect, or not intersect, with the geometries of y.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
mask(x, mask, inverse=FALSE, maskvalues=NA,
      updatevalue=NA, filename="", ...)
```

```
## S4 method for signature 'SpatRaster,SpatVector'
mask(x, mask, inverse=FALSE, updatevalue=NA,
      touches=TRUE, filename="", ...)
```

```
## S4 method for signature 'SpatVector,SpatVector'
mask(x, mask, inverse=FALSE)
```

Arguments

x	SpatRaster or SpatVector
mask	SpatRaster or SpatVector
inverse	logical. If TRUE, areas on mask that are <i>_not_</i> the maskvalue are masked
maskvalues	numeric. The value(s) in mask that indicate which cells of x should be masked (change their value to updatevalue (default = NA))
updatevalue	numeric. The value that masked cells should become (if they are not NA)
touches	logical. If TRUE, all cells touched by lines or polygons will be masked, not just those on the line render path, or whose center point is within the polygon
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[crop](#)

Examples

```
r <- rast(ncols=10, nrows=10)
m <- rast(ncols=10, nrows=10)
values(r) <- 1:100
set.seed(1965)
x <- round(3 * runif(ncell(r)))
x[x==0] <- NA
values(m) <- x
mr <- mask(r, m)
```

match	<i>Value matching for SpatRasters</i>
-------	---------------------------------------

Description

`match` returns a `SpatRaster` with the position of the matched values. The cell values are the index of the table argument.

`%in%` returns a 0/1 (FALSE/TRUE) `SpatRaster` indicating if the cells values were matched or not.

Usage

```
match(x, table, nomatch = NA_integer_, incomparables = NULL)
```

```
x %in% table
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>table</code>	vector of the values to be matched against
<code>nomatch</code>	the value to be returned in the case when no match is found. Note that it is coerced to integer
<code>incomparables</code>	a vector of values that cannot be matched. Any value in <code>x</code> matching a value in this vector is assigned the <code>nomatch</code> value. For historical reasons, FALSE is equivalent to NULL

Value

`SpatRaster`

See Also

[app](#), [match](#)

Examples

```
r <- rast(nrows=10, ncols=10)
values(r) <- 1:100
m <- match(r, c(5:10, 50:55))
n <- r %in% c(5:10, 50:55)
```

Description

Standard mathematical methods for computations with `SpatRaster` objects. Computations are local (applied on a cell by cell basis). If multiple `SpatRaster` objects are used, these must have the same extent and resolution. These have been implemented:

`abs`, `sign`, `sqrt`, `ceiling`, `floor`, `trunc`, `cummax`, `cummin`, `cumprod`, `cumsum`, `log`, `log10`, `log2`, `log1p`, `acos`, `acosh`, `asin`, `asinh`, `atan`, `atanh`, `exp`, `expm1`, `cos`, `cosh`, `sin`, `sinh`, `tan`, `tanh`, `round`, `signif`

Instead of directly calling these methods, you can also provide their name to the `math` method. This is useful if you want to provide an output filename.

The following methods have been implemented for `SpatExtent`: `round`, `floor`, `ceiling`
`round` has also been implemented for `SpatVector`, to round the coordinates of the geometries.

Usage

```
## S4 method for signature 'SpatRaster'
sqrt(x)

## S4 method for signature 'SpatRaster'
log(x, base=exp(1))

## S4 method for signature 'SpatRaster'
round(x, digits=0)

## S4 method for signature 'SpatRaster'
math(x, fun, digits=0, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatVector'
round(x, digits=4)

## S4 method for signature 'SpatRaster'
cumsum(x)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>base</code>	a positive or complex number: the base with respect to which logarithms are computed
<code>digits</code>	Number of digits for rounding
<code>fun</code>	character. Math function name
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>...</code>	additional arguments for writing files as in writeRaster

Value

SpatRaster or SpatExtent

See Also

See [app](#) to use mathematical functions not implemented by the package, and [Arith-methods](#) for arithmetical operations. Use [roll](#) for rolling functions.

Examples

```
r1 <- rast(ncols=10, nrows=10)
v <- runif(ncell(r1))
v[10:20] <- NA
values(r1) <- v
r2 <- rast(r1)
values(r2) <- 1:ncell(r2) / ncell(r2)
r <- c(r1, r2)

s <- sqrt(r)
# same as
math(r, "sqrt")

round(s, 1)

cumsum(r)
```

mem

Memory available and needed

Description

mem_info prints the amount of RAM that is required and available to process a SpatRaster.

free_RAM returns the amount of RAM that is available

Usage

```
mem_info(x, n=1)
```

```
free_RAM()
```

Arguments

x	SpatRaster
n	positive integer. The number of copies of x that are needed

Value

free_RAM returns the amount of available RAM in kilobytes

Examples

```
mem_info(rast())

free_RAM()
```

merge

Merge SpatRasters, or merge a SpatVector with a data.frame

Description

Merge multiple SpatRasters to create a new SpatRaster object with a larger spatial extent. The SpatRasters must have the same origin and spatial resolution. In areas where the SpatRasters overlap, the values of the SpatRaster that is first in the sequence of arguments (or in the SpatRasterCollection) will be retained (unless `first=FALSE`. Cells that are NA are ignored unless `na.rm=FALSE`

There is also a method for merging SpatVector with a data.frame; that is, to join the data.frame to the attribute table of the SpatVector.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
merge(x, y, ..., first=TRUE, na.rm=TRUE, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterCollection,missing'
merge(x, first=TRUE, na.rm=TRUE, filename="", ...)

## S4 method for signature 'SpatVector,data.frame'
merge(x, y, ...)
```

Arguments

<code>x</code>	SpatRaster, SpatRasterCollection, or SpatVector
<code>y</code>	missing if <code>x</code> is a SpatRasterCollection. SpatRaster if <code>x</code> is a SpatRaster. data.frame if <code>x</code> is a SpatVector
<code>...</code>	if <code>x</code> is a SpatRaster: additional objects of the same class as <code>x</code> . If <code>x</code> is a SpatRasterCollection: options for writing files as in writeRaster . If <code>x</code> is a SpatVector, the same arguments as in merge
<code>first</code>	logical. If TRUE, in areas where rasters overlap, the first value is used. Otherwise the last value is used
<code>na.rm</code>	logical. If TRUE values that are NA are ignored
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster

Value

SpatRaster or SpatVector

See Also

Combining tiles with [vrt](#) may be more efficient than using merge. See [mosaic](#) for averaging overlapping regions.

See [classify](#) to merge a SpatRaster and a data.frame and [union](#) to combine SpatExtent objects.

Examples

```
x <- rast(xmin=-110, xmax=-80, ymin=40, ymax=70, res=1, vals=1)
y <- rast(xmin=-85, xmax=-55, ymax=60, ymin=30, res=1, vals=2)
z <- rast(xmin=-60, xmax=-30, ymax=50, ymin=20, res=1, vals=3)

m1 <- merge(x, y, z)
m2 <- merge(z, y, x)
m3 <- merge(y, x, z)

# if you have many SpatRasters, it may be convenient
# to make a SpatRasterCollection
rlist <- list(x, y, z)
rsrc <- sprc(rlist)

m <- merge(rsrc)

## SpatVector with data.frame
f <- system.file("ex/lux.shp", package="terra")
p <- vect(f)
dfr <- data.frame(District=p$NAME_1, Canton=p$NAME_2, Value=round(runif(length(p), 100, 1000)))
dfr <- dfr[1:5, ]
pm <- merge(p, dfr, all.x=TRUE, by.x=c('NAME_1', 'NAME_2'), by.y=c('District', 'Canton'))
pm
values(pm)
```

mergeTime

merge SpatRasters by timelines to create a single timeseries

Description

Combine SpatRasters with partly overlapping time-stamps to create a single time series. If there is no overlap between the SpatRasters there is no point in using this function (use [c](#) instead).

Also note that time gaps are not filled. You can use [fillTime](#) to do that.

Usage

```
## S4 method for signature 'SpatRasterDataset'
mergeTime(x, fun=mean, filename="", ...)
```

Arguments

x	SpatRasterDataset
fun	A function that reduces a vector to a single number, such as mean or min
filename	character. Output filename
...	list with named options for writing files as in writeRaster

Value

SpatRaster

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
s1 <- c(r, r)
time(s1) <- as.Date("2001-01-01") + 0:5
s1 <- s1/10
time(s1) <- as.Date("2001-01-07") + 0:5
s2 <- s1*10
time(s2) <- as.Date("2001-01-05") + 0:5
x <- sds(s1, s1, s2)

m <- mergeTime(x, mean)
```

meta	<i>meta</i>
------	-------------

Description

Get metadata associated with the sources or layers of a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
meta(x, layers=FALSE)
```

Arguments

x	SpatRaster
layers	logical. Should the layer level metadata be returned?

Value

list

modal	<i>modal value</i>
-------	--------------------

Description

Compute the mode for each cell across the layers of a `SpatRaster`. The mode, or modal value, is the most frequent value in a set of values.

Usage

```
## S4 method for signature 'SpatRaster'  
modal(x, ..., ties="first", na.rm=FALSE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>...</code>	additional argument of the same type as <code>x</code> or numeric
<code>ties</code>	character. Indicates how to treat ties. Either "random", "lowest", "highest", "first", or "NA"
<code>na.rm</code>	logical. If TRUE, NA values are ignored. If FALSE, NA is returned if <code>x</code> has any NA values
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster

Value

`SpatRaster`

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))  
r <- c(r/2, r, r*2)  
m <- modal(r)
```

mosaic

*mosaic SpatRasters***Description**

Combine adjacent and (partly) overlapping SpatRasters to form a single new SpatRaster. Values in overlapping cells are averaged (by default) or can be computed with another function.

The SpatRasters must have the same origin and spatial resolution.

This method is similar to the simpler, but much faster, [merge](#) method.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
mosaic(x, y, ..., fun="mean", filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterCollection,missing'
mosaic(x, fun="mean", filename="", ...)
```

Arguments

x	SpatRaster
y	object of same class as x
...	additional SpatRasters
fun	character. One of "sum", "mean", "median", "min", "max"
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

[merge](#)

Examples

```
x <- rast(xmin=-110, xmax=-60, ymin=40, ymax=70, res=1, vals=1)
y <- rast(xmin=-95, xmax=-45, ymax=60, ymin=30, res=1, vals=2)
z <- rast(xmin=-80, xmax=-30, ymax=50, ymin=20, res=1, vals=3)

m1 <- mosaic(x, y, z)

m2 <- mosaic(z, y, x)
```



```
# with many SpatRasters, make a SpatRasterCollection from a list
rlist <- list(x, y, z)
rsrc <- sprc(rlist)

m <- mosaic(rsrc)
```

na.omit	<i>na.omit for SpatVector</i>
---------	-------------------------------

Description

Remove empty geometries and/or records that are NA from a SpatVector.

Usage

```
## S4 method for signature 'SpatVector'
na.omit(object, field=NA, geom=FALSE)
```

Arguments

object	SpatVector
field	character or NA. If NA, missing values in the attributes are ignored. Other values are either one or more field (variable) names, or "" to consider all fields
geom	logical. If TRUE empty geometries are removed

Value

SpatVector

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$test <- c(1,2,NA)
nrow(v)
x <- na.omit(v, "test")
nrow(x)
```

NAflag

Set the NA flag

Description

The main purpose of this method is to allow correct reading of a SpatRaster that is based on a file that has an incorrect NA flag. The file is not changed, but flagged value is set to NA when values are read from the file ("lazy evaluation"). In contrast, if the values are in memory the change is made immediately.

To change values, it is generally better to use [classify](#)

Usage

```
## S4 method for signature 'SpatRaster'
NAflag(x)

## S4 replacement method for signature 'SpatRaster'
NAflag(x)<-value
```

Arguments

x	SpatRaster
value	numeric. The value to be interpreted as NA; set this before reading the values from the file. This can be a single value, or multiple values, one for each data source (file / subdataset)

Value

none or numeric

See Also

[classify](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))[[1]]
NAflag(s) <- 255
plot(s)
NAflag(s)
```

names	<i>Names of Spat* objects</i>
-------	-------------------------------

Description

Get or set the names of the layers of a `SpatRaster` or the attributes of a `SpatVector`. With `longnames` you can get or set the "long names" of a `SpatRaster` or `SpatRasterDataset`.

For a `SpatRaster`, you can also get/set a variable name or long name (one per data source).

See [set.names](#) for in-place setting of names.

Usage

```
## S4 method for signature 'SpatRaster'
names(x)

## S4 replacement method for signature 'SpatRaster'
names(x)<-value

## S4 method for signature 'SpatRaster'
varnames(x)

## S4 replacement method for signature 'SpatRaster'
varnames(x)<-value

## S4 method for signature 'SpatRaster'
longnames(x)

## S4 replacement method for signature 'SpatRaster'
longnames(x)<-value

## S4 method for signature 'SpatRasterDataset'
names(x)

## S4 replacement method for signature 'SpatRasterDataset'
names(x)<-value

## S4 method for signature 'SpatRasterDataset'
varnames(x)

## S4 replacement method for signature 'SpatRasterDataset'
varnames(x)<-value

## S4 method for signature 'SpatRasterDataset'
longnames(x)
```

```
## S4 replacement method for signature 'SpatRasterDataset'
longnames(x)<-value

## S4 method for signature 'SpatVector'
names(x)

## S4 replacement method for signature 'SpatVector'
names(x)<-value
```

Arguments

x	SpatRaster, SpatRasterDataset, or SpatVector
value	character (vector)

Value

character

Note

terra enforces neither unique nor valid names. See [make.unique](#) to create unique names and `{make.names}` to make syntactically valid names.

Examples

```
s <- rast(ncols=5, nrows=5, nlyrs=3)
nlyr(s)
names(s)
names(s) <- c("a", "b", "c")
names(s)

# space is not valid
names(s)[2] <- "hello world"
names(s)

# two invalid names
names(s) <- c("a", " a ", "3")
names(s)

# SpatVector names
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
names(v)
names(v) <- paste0(substr(names(v), 1, 2), "_", 1:ncol(v))
names(v)
```

nearest	<i>nearby geometries</i>
---------	--------------------------

Description

Identify geometries that are near to each other. Either get the index of all geometries within a certain distance, or the k nearest neighbors, or (with nearest) get the nearest points between two geometries.

Usage

```
## S4 method for signature 'SpatVector'
nearby(x, y=NULL, distance=0, k=1, centroids=TRUE, symmetrical=TRUE)
```

```
## S4 method for signature 'SpatVector'
nearest(x, y, pairs=FALSE, centroids=TRUE, lines=FALSE)
```

Arguments

x	SpatVector
y	SpatVector or NULL
distance	numeric. maximum distance
k	positive integer. number of neighbors. Ignored if distance > 0
centroids	logical. Should the centroids of polygons be used?
symmetrical	logical. If TRUE, a near pair is only included once. That is, if geometry 1 is near to geometry 3, the implied nearness between 3 and 1 is not reported. Ignored if k neighbors are returned
pairs	logical. If TRUE pairwise nearest points are returned (only relevant when using at least one SpatVector of lines or polygons)
lines	logical. If TRUE lines between the nearest points instead of (the nearest) points

Value

matrix

See Also

[relate](#), [adjacent](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
nearby(v, distance=12000)
```

<code>normalize.longitude</code>	<i>normalize vector data that crosses the dateline</i>
----------------------------------	--

Description

Normalize the longitude of geometries, move them if they are outside of the -180 to 180 degrees range.

Usage

```
## S4 method for signature 'SpatVector'
normalize.longitude(x)
```

Arguments

x SpatVector

Value

SpatVector

See Also

[rotate](#) for SpatRaster

Examples

```
p <- vect("POLYGON ((120 10, 230 75, 230 -75, 120 10))")
normalize.longitude(p)
```

<code>north</code>	<i>North arrow</i>
--------------------	--------------------

Description

Add a (North) arrow to a map

Usage

```
north(xy=NULL, type=1, label="N", angle=0, d, head=0.1, xpd=TRUE, ...)
```

Arguments

xy	numeric. x and y coordinate to place the arrow. It can also be one of following character values: "bottomleft", "bottom", "bottomright", "topleft", "top", "topright", "left", "right", or NULL
type	integer between 1 and 12, or a character (unicode) representation of a right pointing arrow such as "\u27A9"
label	character, to be printed near the arrow
angle	numeric. The angle of the arrow in degrees
d	numeric. Distance covered by the arrow in plot coordinates. Only applies to type=1
head	numeric. The size of the arrow "head", for type=1
xpd	logical. If TRUE, the scale bar or arrow can be outside the plot area
...	graphical arguments to be passed to other methods

Value

none

See Also

[sbar](#), [plot](#), [inset](#)

Examples

```
f <- system.file("ex/meuse.tif", package="terra")
r <- rast(f)
plot(r)
north()
north(c(178550, 332500), d=250)

## Not run:
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r, type="interval")
sbar(15, c(6.3, 50), type="bar", below="km", label=c(0,7.5,15), cex=.8)
north(type=3, cex=.8)
north(xy=c(6.7, 49.9), type=2, angle=45, label="NE")
north(xy=c(6.6, 49.7), type=5, cex=1.25)
north(xy=c(5.5, 49.6), type=9)
north(d=.05, xy=c(5.5, 50), angle=180, label="S", lwd=2, col="blue")

## all arrows
r <- rast(res=10)
values(r) <- 1
plot(r, col="white", axes=FALSE, legend=FALSE, mar=c(0,0,0,0), reset=TRUE)
for (i in 1:12) {
  x = -200+i*30
  north(xy=cbind(x,30), type=i)
```

```
text(x, -20, i, xpd=TRUE)
}
```

```
## End(Not run)
```

not.na	<i>is not NA</i>
--------	------------------

Description

Shortcut method to avoid the two-step `!is.na(x)`

Usage

```
## S4 method for signature 'SpatRaster'
not.na(x, falseNA, filename="", ...)
```

Arguments

x	SpatRaster
falseNA	logical. Should the result be TRUE, NA instead of TRUE, FALSE?
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[Compare-methods](#)

Examples

```
r <- rast(ncols=10, nrows=10, vals=1)
r[10:20] <- NA
x <- not.na(r)
```

options	<i>Options</i>
---------	----------------

Description

Get or set general options.

Usage

```
terraOptions(..., print=TRUE)
```

Arguments

...	option names and values (see Details). Or missing, to get or show the current options
print	logical. If TRUE the option names and values are printed

Details

The following options are available.

memfrac - value between 0 and 0.9 (larger values give a warning). The fraction of RAM that may be used by the program.

memmin - if memory required is below this threshold (in GB), the memory is assumed to be available. Otherwise, terra checks if it is available.

memmax - the maximum amount of RAM (in GB) that terra is allowed to use when processing a raster dataset. Should be less than what is detected (see [mem_info](#)), and higher values are ignored. Set it to a negative number or NA to not set this option. `terraOptions` only shows the value of `memmax` if it is set.

tempdir - directory where temporary files are written. The default what is returned by `tempdir()`.

datatype - default data type. See [writeRaster](#)

todisk - logical. If TRUE write all raster data to disk (temp file if no file name is specified). For debugging.

progress - non-negative integer. A progress bar is shown if the number of chunks in which the data is processed is larger than this number. No progress bar is shown if the value is zero

verbose - logical. If TRUE debugging info is printed for some functions

Value

list. Invisibly if `print=TRUE`

Note

It is possible to set your own default options in "etc/.Rprofile.site" of your R installation like this

```
options(terra_default=list(tempdir="d:/temp", memfrac=.4))
```

But that may not be a good practise. It is clearer to set your favorite options at the beginning of each script.

Examples

```
terraOptions()
terraOptions(memfrac=0.5, tempdir = "c:/temp")
terraOptions(progress=10)
terraOptions()
```

origin	<i>Origin</i>
--------	---------------

Description

Get or set the coordinates of the point of origin of a SpatRaster. This is the point closest to (0, 0) that you could get if you moved towards that point in steps of the x and y resolution.

Usage

```
## S4 method for signature 'SpatRaster'
origin(x)

## S4 replacement method for signature 'SpatRaster'
origin(x)<-value
```

Arguments

x	SpatRaster
value	numeric vector of length 1 or 2

Value

A vector of two numbers (x and y coordinates)

Examples

```
r <- rast(xmin=-0.5, xmax = 9.5, ncols=10)
origin(r)
origin(r) <- c(0,0)
r
```

pairs	<i>Pairs plot (matrix of scatterplots)</i>
-------	--

Description

Pair plots of layers in a SpatRaster. This is a wrapper around graphics function [pairs](#).

Usage

```
## S4 method for signature 'SpatRaster'
pairs(x, hist=TRUE, cor=TRUE, use="pairwise.complete.obs", maxcells=100000, ...)
```

Arguments

x	SpatRaster
hist	logical. If TRUE a histogram of the values is shown on the diagonal
cor	logical. If TRUE the correlation coefficient is shown in the upper panels
use	argument passed to the cor function
maxcells	integer. Number of pixels to sample from each layer of a large SpatRaster
...	additional arguments (graphical parameters)

See Also

[boxplot](#), [hist](#)

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
s <- c(r, 1/r, sqrt(r))
names(s) <- c("elevation", "inverse", "sqrt")
pairs(s)

# to make individual histograms:
hist(r)
# or scatter plots:
plot(s[[1]], s[[2]])
```

panel

Map panel

Description

Show multiple maps that share a single legend.

Usage

```
## S4 method for signature 'SpatRaster'
panel(x, main, loc.main="topleft", nc, nr, maxnl=16,
maxcell=500000, box=FALSE, pax=list(), plg=list(), ...)
```

Arguments

x	SpatRaster
main	character. Main plot titles (one for each layer to be plotted). You can use arguments <code>cex.main</code> , <code>font.main</code> , <code>col.main</code> to change the appearance
loc.main	numeric of character to set the location of the main title. Either two coordinates, or a character value such as "topleft")
nc	positive integer. Optional. The number of columns to divide the plotting device in (when plotting multiple layers)
nr	positive integer. Optional. The number of rows to divide the plotting device in (when plotting multiple layers)
maxnl	positive integer. Maximum number of layers to plot (for a multi-layer object)
maxcell	positive integer. Maximum number of cells to use for the plot
box	logical. Should a box be drawn around the map?
plg	see plot
pax	see plot
...	arguments passed to <code>plot("SpatRaster", "numeric")</code> and additional graphical arguments

See Also

[plot](#) and see `rasterVis::levelplot` and `tidyterra::autoplot` for more sophisticated panel plots.

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
v <- vect(system.file("ex/lux.shp", package="terra"))
x <- c(r, r/2, r*2, r)
names(x) <- paste0("(", LETTERS[1:4], ")")
panel(x)
panel(x, fun=\()lines(v), loc.main="topright")
```

patches	<i>Detect patches (clumps) of cells</i>
---------	---

Description

Detect patches (clumps). Patches are groups of cells that are surrounded by cells that are NA. Set zeroAsNA to TRUE to also identify patches separated by cells with values of zero.

Usage

```
## S4 method for signature 'SpatRaster'
patches(x, directions=4, zeroAsNA=FALSE, allowGaps=TRUE, filename="", ...)
```

Arguments

x	SpatRaster
directions	integer indicating which cells are considered adjacent. Should be 8 (Queen's case) or 4 (Rook's case)
zeroAsNA	logical. If TRUE treat cells that are zero as if they were NA
allowGaps	logical. If TRUE there may be gaps in the patch IDs (e.g. you may have patch IDs 1, 2, 3 and 5, but not 4). If it is FALSE, these numbers will be recoded from 1 to the number of patches (4 in this example)
filename	character. Output filename
...	options for writing files as in writeRaster

Value

SpatRaster. Cell values are patch numbers

See Also

[focal](#), [boundaries](#)

Examples

```
r <- rast(nrows=18, ncols=36, xmin=0)
r[1:2, 5:8] <- 1
r[5:8, 2:6] <- 1
r[7:12, 22:36] <- 1
r[15:16, 18:29] <- 1
p <- patches(r)

# zero as background instead of NA
r <- rast(nrows=10, ncols=10, xmin=0, vals=0)
r[3, 3] <- 10
r[4, 4] <- 10
r[5, 5:8] <- 12
```

```

r[6, 6:9] <- 12

# treat zeros as NA

p4 <- patches(r, zeroAsNA=TRUE)
p8 <- patches(r, 8, zeroAsNA=TRUE)

### patches for different values
# remove zeros manually
rr <- classify(r, cbind(0, NA))

# make layers for each value
s <- segregate(rr, keep=TRUE, other=NA)
p <- patches(s)

### patch ID values are not guaranteed to be consecutive
r <- rast(nrows=5, ncols=10, xmin=0)
set.seed(0)
values(r)<- round(runif(ncell(r))*0.7)
rp <- patches(r, directions=8, zeroAsNA=TRUE)
plot(rp, type="classes"); text(rp)

## unless you set allowGaps=FALSE
rp <- patches(r, directions=8, zeroAsNA=TRUE, allowGaps=FALSE)
plot(rp, type="classes"); text(rp)

### use zonal to remove small patches
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- classify(r, cbind(-Inf, 400, NA))
y <- patches(x)
# remove patches smaller than 100 ha
rz <- zonal(cellSize(y, unit="ha"), y, sum, as.raster=TRUE)
s <- ifel(rz < 250, NA, y)

```

perim

Perimeter or length

Description

This method returns the length of lines or the perimeter of polygons.

When the crs is not longitude/latitude, you may get more accurate results by first unprojecting the *SpatVector* (you can use [project](#) to transform the crs to longitude/latitude)

Usage

```

## S4 method for signature 'SpatVector'
perim(x)

```

Arguments

x SpatVector

Value

numeric (m)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
perim(v)
```

persp

Perspective plot

Description

Perspective plot of a SpatRaster. This is an implementation of a generic function in the graphics package.

Usage

```
## S4 method for signature 'SpatRaster'
persp(x, maxcells=100000, ...)
```

Arguments

x SpatRaster. Only the first layer is used

maxcells integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), spatSample(method="regular") is used before plotting

... Any argument that can be passed to [persp](#) (graphics package)

See Also

[persp](#), [contour](#), [plot](#)

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
persp(r)
```

plet

*Plot with leaflet***Description**

Plot the values of a `SpatRaster` or `SpatVector` to make an interactive leaflet map that is displayed in a browser.

These methods require the development version of leaflet that you can install with `remotes::install_github("rstudio/leaflet")`.

Usage

```
## S4 method for signature 'SpatRaster'
plet(x, y=1, col, alpha=0.8, main=names(x), tiles=NULL,
     wrap=TRUE, maxcell=500000, legend="bottomright",
     shared=FALSE, panel=FALSE, collapse=TRUE, map=NULL)

## S4 method for signature 'SpatVector'
plet(x, y="", col, alpha=1, fill=0, main=y, cex=1, lwd=2, popup=TRUE,
     label=FALSE, split=FALSE, tiles=c("Streets", "Esri.WorldImagery", "OpenTopoMap"),
     wrap=TRUE, legend="bottomright", collapse=FALSE, map=NULL, ...)

## S4 method for signature 'SpatVectorCollection'
plet(x, col, alpha=1, fill=0, cex=1, lwd=2, popup=TRUE,
     label=FALSE, tiles=c("Streets", "Esri.WorldImagery", "OpenTopoMap"), wrap=TRUE,
     legend="bottomright", collapse=FALSE, map=NULL)

## S4 method for signature 'leaflet'
lines(x, y, col, lwd=2, alpha=1, ...)

## S4 method for signature 'leaflet'
points(x, y, col, cex=1, alpha=1, popup=FALSE, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code> , <code>SpatVector</code> , or leaflet object
<code>y</code>	missing, or positive integer, or character (variable or layer name) indicating the layer(s) to be plotted. If <code>x</code> is a <code>SpatRaster</code> , you can select multiple layers
<code>col</code>	character. Vector of colors or color generating function
<code>alpha</code>	Number between 0 and 1 to set the transparency for lines (0 is transparent, 1 is opaque)
<code>fill</code>	Number between 0 and 1 to set the transparency for polygon areas (0 is transparent, 1 is opaque)
<code>tiles</code>	character or NULL. Names of background tile providers

wrap	logical. if TRUE, tiles wrap around
maxcell	positive integer. Maximum number of cells to use for the plot
legend	character to indicate the legend position ("bottomleft", "bottomright", "topleft" or "topright") or NULL to suppress the legend
main	character. Title for the legend. The length should be 1 if x is a SpatVector and length nlyr(x) if x is a SpatVector
shared	logical. Should the legend be the same for all rasters (if multiple layers of SpatRaster x are mapped)
map	leaflet object
...	additional arguments for drawing points, lines, or polygons passed on the the relevant leaflet function
collapse	logical. Should the layers "control" panel be collapsed?
split	logical. IF TRUE a check-box is created to toggle each value in codey (If x is a SpatVector)
cex	numeric. point size magnifier. See par
lwd	numeric, line-width. See par
popup	logical. Should pop-ups be created?
label	logical. Should mouse-over labels be added?
panel	logical. Should SpatRaster layers be shown as a panel"

See Also

[plot](#)

Examples

```
## Not run:
if (require(leaflet)) {
  if (packageVersion("leaflet") > "2.1.1") {

    v <- vect(system.file("ex/lux.shp", package="terra"))
    p <- spatSample(as.polygons(v, ext=T), 30)
    values(p) = data.frame(id=11:40, name=letters[1:30])

    m <- plet(v, "NAME_1", alpha=.4, tiles="")
    m <- points(m, p, col="red", cex=2, popup=T)
    lines(m, v)

    plet(v, "NAME_1", split=TRUE, alpha=.2) |>
      points(p, col="gray", cex=2, popup=TRUE,
        clusterOptions = markerClusterOptions()) |> lines(v)

    s <- svc(v, p)
    names(s) <- c("the polys", "set of points")
    plet(s, col=c("red", "blue"), lwd=1)
```

```

r <- rast(system.file("ex/elev.tif", package="terra"))
plot(r, main="Hi\nthere") |> lines(v, lwd=1)

plot(r, tiles="Streets") |> lines(v, lwd=2, col="blue")

x <- c(r, 50*classify(r, 5))
names(x) <- c("first", "second")

# each their own legend
plot(x, 1:2, tiles="Streets", collapse=FALSE) |> lines(v, lwd=2, col="blue")

# shared legend
plot(x, 1:2, tiles="Streets", shared=TRUE, collapse=FALSE) |> lines(v, lwd=2, col="blue")

}}
## End(Not run)

```

plot

Make a map

Description

Plot the values of a `SpatRaster` or `SpatVector` to make a map.

See [points](#), [lines](#) or [polys](#) to add a `SpatVector` to an existing map (or use argument `add=TRUE`).

There is a separate help file for plotting a [SpatGraticule](#) or [SpatExtent](#).

Usage

```

## S4 method for signature 'SpatRaster,numeric'
plot(x, y=1, col, type, mar=NULL, legend=TRUE, axes=TRUE, plg=list(), pax=list(),
      maxcell=500000, smooth=FALSE, range=NULL, levels=NULL, all_levels=FALSE,
      breaks=NULL, breakby="eqint", fun=NULL, colNA=NULL, alpha=NULL, sort=FALSE,
      decreasing=FALSE, grid=FALSE, ext=NULL, reset=FALSE, add=FALSE, buffer=FALSE,
      background=NULL, box=axes, clip=TRUE, ...)

## S4 method for signature 'SpatRaster,missing'
plot(x, y, main, mar=NULL, nc, nr, maxnl=16, maxcell=500000, ...)

## S4 method for signature 'SpatRaster,character'
plot(x, y, ...)

## S4 method for signature 'SpatVector,character'
plot(x, y, col=NULL, type, mar=NULL, add=FALSE, legend=!add, axes=!add,
      main="", buffer=TRUE, background=NULL, grid=FALSE, ext=NULL, sort=TRUE,
      decreasing=FALSE, plg=list(), pax=list(), nr, nc, colNA=NA, alpha=NULL,
      box=axes, clip=TRUE, ...)

```

```
## S4 method for signature 'SpatVector,numeric'
plot(x, y, ...)
```

```
## S4 method for signature 'SpatVector,missing'
plot(x, y, values=NULL, ...)
```

Arguments

<code>x</code>	SpatRaster or SpatVector
<code>y</code>	missing or positive integer or name indicating the layer(s) to be plotted
<code>col</code>	character. Colors. The default is <code>rev(grDevices::terrain.colors(50))</code> . If <code>x</code> is a SpatRaster, it can also be a data.frame with two columns (value, color) to get a "classes" type legend or with three columns (from, to, color) to get an "interval" type legend
<code>type</code>	character. Type of map/legend. One of "continuous", "classes", or "interval". If not specified, the type is chosen based on the data
<code>mar</code>	numeric vector of length 4 to set the margins of the plot (to make space for the legend). The default is (3.1, 3.1, 2.1, 7.1) for a single plot with a legend and (3.1, 3.1, 2.1, 2.1) otherwise. Use <code>mar=NA</code> to not set the margins
<code>legend</code>	logical or character. If not FALSE a legend is drawn. The character value can be used to indicate where the legend is to be drawn. For example "topright" or "bottomleft". Use <code>p1g</code> for more refined placement. Not supported for continuous legends (the default for raster data)
<code>axes</code>	logical. Draw axes?
<code>buffer</code>	logical. If TRUE the plotting area is made slightly larger than the extent of <code>x</code>
<code>background</code>	background color. Default is no color (white)
<code>box</code>	logical. Should a box be drawn around the map?
<code>clip</code>	logical. Should the axes be clipped to the extent of <code>x</code> ?
<code>p1g</code>	list with parameters for drawing the legend. See the arguments for legend . Parameter <code>digits</code> can be used to set the number of digits to print (after the decimal point for a continuous legend). Parameter <code>size</code> can be used to change the height and/or width of a continuous legend. The defaults are <code>c(1,1)</code> , negative values for <code>size</code> flip the order of the legend
<code>pax</code>	list with parameters for drawing axes. See the arguments for axis . Arguments <code>side</code> , <code>tick</code> and <code>lab</code> can be used to indicate for which of the four axes to draw a line (side), tick-mark, and/or the tick-mark labels. The default is <code>c(1:4)</code> for <code>side</code> and <code>1:2</code> for the other two. If <code>side</code> is changed the other two default to that value. Logical argument <code>retro</code> can be used to use a sexagesimal notation for the labels (degrees/minutes/hemisphere) instead of the standard decimal notation
<code>maxcell</code>	positive integer. Maximum number of cells to use for the plot
<code>smooth</code>	logical. If TRUE the cell values are smoothed (only if a continuous legend is used)
<code>range</code>	numeric. minimum and maximum values to be used for the continuous legend
<code>levels</code>	character. labels for the legend when <code>type="classes"</code>

<code>all_levels</code>	logical. If TRUE, the legend shows all levels of a categorical raster, even if they are not present in the data
<code>breaks</code>	numeric. Either a single number to indicate the number of breaks desired, or the actual breaks. When providing this argument, the default legend becomes "interval"
<code>breakby</code>	character or function. Either "eqint" for equal interval breaks, "cases" for equal quantile breaks. If a function is supplied it should take a single argument (a vector of values) and create groups
<code>fun</code>	function to be called after plotting each SpatRaster layer to add something to each map (such as text, legend, lines). For example, with SpatVector <code>v</code> , you could do <code>fun=function() lines(v)</code> . The function may have one argument, representing the layer that is plotted (1 to the number of layers)
<code>colNA</code>	character. color for the NA values
<code>alpha</code>	Either a single numeric between 0 and 1 to set the transparency for all colors (0 is transparent, 1 is opaque) or a SpatRaster with values between 0 and 1 to set the transparency by cell. To set the transparency for a given color, set it to the colors directly
<code>sort</code>	logical. If TRUE legends with categorical values are sorted. If <code>x</code> is a SpatVector you can also supply a vector of the unique values, in the order in which you want them to appear in the legend
<code>decreasing</code>	logical. If TRUE, legends are sorted in decreasing order
<code>grid</code>	logical. If TRUE grid lines are drawn. Their properties such as type and color can be set with the <code>pax</code> argument
<code>nc</code>	positive integer. Optional. The number of columns to divide the plotting device in (when plotting multiple layers)
<code>nr</code>	positive integer. Optional. The number of rows to divide the plotting device in (when plotting multiple layers)
<code>main</code>	character. Main plot titles (one for each layer to be plotted). You can use arguments <code>cex.main</code> , <code>font.main</code> , <code>col.main</code> to change the appearance; and <code>loc.main</code> to change the location of the main title (either two coordinates, or a character value such as "topleft")
<code>maxnl</code>	positive integer. Maximum number of layers to plot (for a multi-layer object)
<code>add</code>	logical. If TRUE add the object to the current plot
<code>ext</code>	SpatExtent. Can be use instead of <code>xlim</code> and <code>ylim</code> to set the extent of the plot
<code>reset</code>	logical. If TRUE add the margins (see argument <code>mar</code>) are reset to what they were before calling <code>plot</code> ; doing so may affect the display of additional objects that are added to the map (e.g. with <code>lines</code>)
<code>values</code>	Either a vector with values to be used for plotting or a two-column data.frame, where the first column matches a variable in <code>x</code> and the second column has the values to be plotted
<code>...</code>	arguments passed to <code>plot("SpatRaster", "numeric")</code> and additional graphical arguments

See Also

[points](#), [lines](#), [polys](#), [image](#), [scatterplot](#), multiple layers: [panel](#), scale bar: [sbar](#), north arrow: [north](#), plot a [SpatGraticule](#) or [SpatExtent](#), [plotRGB](#)

Examples

```
## SpatRaster
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r)

plot(r, type="interval")

e <- c(6.37, 6.41, 49.9, 50.1)
plot(r, plg=list(ext=e, title="Legend\nTitle", title.cex=0.9),
pax=list(side=1:4, retro=TRUE))
north(cbind(5.8, 50.1))

d <- classify(r, c(100,200,300,400,500,600))
plot(d, type="classes")

plot(d, type="interval", breaks=1:5)
plot(d, type="interval", breaks=c(1,4,5), plg=list(legend=c("1-4", "4-5")))
plot(d, type="classes", xlim=c(5.6, 6.6),
plg=list(legend=c("Mr", "Xx", "As", "Zx", "Bb"), x="bottomleft"))

x <- trunc(r/200)
levels(x) <- data.frame(id=0:2, element=c("earth", "wind", "fire"))
plot(x, plg=list(x="topright"),mar=c(2,2,2,2))

oldpar <- par(no.readonly=TRUE)

# two plots with the same legend
dev.new(width=6, height=4, noRStudioGD = TRUE)
par(mfrow=c(1,2))
plot(r, range=c(50,600), mar=c(1,1,1,4))
plot(r/2, range=c(50,600), mar=c(1,1,1,4))

# as we only need one legend:
par(mfrow=c(1,2))
plot(r, range=c(50,600), mar=c(2, 2, 2, 2), plg=list(size=0.9, cex=.8),
pax=list(side=1:2, cex.axis=.6), box=FALSE)
#text(182500, 335000, "Two maps, one plot", xpd=NA)
plot(r/2, range=c(50,600), mar=c(2, 2, 2, 2), legend=FALSE,
pax=list(side=c(1,4), cex.axis=.6), box=FALSE)

par(oldpar)

# multi-layer with RGB
s <- rast(system.file("ex/logo.tif", package="terra"))
```

```

s
plot(s)
# remove RGB
plot(s*1)
# or use layers
plot(s, 1)
plot(s, 1:3)

# fix legend by linking values and colors

x = rast(nrows = 2, ncols = 2, vals=1)
y = rast(nrows = 2, ncols = 2, vals=c(1,2,2,1))
cols = data.frame(id=1:2, col=c("red", "blue"))
plot(c(x,y), col=cols)

r = rast(nrows=10, ncols=10, vals=1:100)
dr = data.frame(from=c(5,33,66,150), to=c(33, 66, 95,200), col=rainbow(4))
plot(r, col=dr)

### SpatVector

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

plot(v)

plot(v, "NAME_2", col=rainbow(12), border=c("gray", "blue"), lwd=3)

plot(v, 2, pax=list(side=1:2), plg=list(x=6.16, y=50.17, cex=.8), xlim=c(5.7, 6.7))

plot(v, 4, pax=list(side=1:2), plg=list(x=6.2, y=50.2, ncol=2), main="", box=FALSE)

plot(v, 1, plg=list(x=5.8, y=49.37, horiz=TRUE, cex=1.1), main="", mar=c(5,2,0.5,0.5))

plot(v, density=1:12, angle=seq(18, 360, 20), col=rainbow(12))

plot(v, "AREA", type="interval", breaks=3, mar=c(3.1, 3.1, 2.1, 3.1),
      plg=list(x="topright"), main="")

plot(v, "AREA", type="interval", breaks=c(0,200,250,350),
      mar=c(2,2,2,2), xlim=c(5.7, 6.75),
      plg=list(legend=c("<200", "200-250", ">250"), cex=1, bty="o",
                x=6.3, y=50.15, box.lwd=2, bg="light yellow", title="My legend"))

```

Description

Make a Red-Green-Blue plot based on three layers in a `SpatRaster`. The layers (sometimes referred to as "bands" because they may represent different bandwidths in the electromagnetic spectrum) are combined such that they represent the red, green and blue channel. This function can be used to make "true" (or "false") color images from Landsat and other multi-spectral satellite images.

Note that the margins of the plot are set to zero (no axes or titles are visible) but can be set with the `mar` argument.

An alternative way to plot RGB images is to first use `colorize` to create a single layer `SpatRaster` with a color-table and then use `plot`.

Usage

```
## S4 method for signature 'SpatRaster'
plotRGB(x, r=1, g=2, b=3, a=NULL, scale, maxcell=500000, mar=0,
stretch=NULL, ext=NULL, smooth=TRUE, colNA="white", alpha, bgalpha,
addfun=NULL, zlim=NULL, zlimcol=NULL, axes=FALSE, xlab="", ylab="",
asp=NULL, add=FALSE, xlim, ylim,...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>r</code>	integer. Index of the Red channel, between 1 and <code>nlyr(x)</code>
<code>g</code>	integer. Index of the Green channel, between 1 and <code>nlyr(x)</code>
<code>b</code>	integer. Index of the Blue channel, between 1 and <code>nlyr(x)</code>
<code>a</code>	integer. Index of the alpha (transparency) channel, between 1 and <code>nlyr(x)</code> . If not <code>NULL</code> , argument <code>alpha</code> is ignored
<code>scale</code>	integer. Maximum (possible) value in the three channels. Defaults to 255 or to the maximum value of <code>x</code> if that is known and larger than 255
<code>maxcell</code>	integer > 0. Maximum number of pixels to use
<code>mar</code>	numeric vector recycled to length 4 to set the margins of the plot. Use <code>mar=NULL</code> or <code>mar=NA</code> to not set the margins
<code>stretch</code>	character. Option to stretch the values to increase contrast: "lin" (linear) or "hist" (histogram)
<code>ext</code>	An <code>SpatExtent</code> object to zoom in to a region of interest (see <code>draw</code>)
<code>smooth</code>	logical. If <code>TRUE</code> , smooth the image when drawing to get the appearance of a higher spatial resolution
<code>colNA</code>	color for the background (NA values)
<code>alpha</code>	transparency. Integer between 0 (transparent) and 255 (opaque)
<code>bgalpha</code>	Background transparency. Integer between 0 (transparent) and 255 (opaque)
<code>addfun</code>	Function to add additional items such as points or polygons to the plot (map). See <code>plot</code>

<code>zlim</code>	numeric vector of length 2. Range of values to plot (optional). If this is set, and <code>stretch="lin"</code> is used, then the values are stretched within the range of <code>zlim</code> . This allows creating consistent coloring between <code>SpatRasters</code> with different cell-value ranges, even when stretching the colors for improved contrast
<code>zlimcol</code>	If <code>NULL</code> the values outside the range of <code>zlim</code> get the color of the extremes of the range. If <code>zlimcol</code> has any other value, the values outside the <code>zlim</code> range get the color of NA values (see <code>colNA</code>)
<code>axes</code>	logical. If <code>TRUE</code> axes are drawn (and arguments such as <code>main="title"</code> will be honored)
<code>xlab</code>	character. Label of x-axis
<code>ylab</code>	character. Label of y-axis
<code>asp</code>	numeric. Aspect (ratio of x and y. If <code>NULL</code> , and appropriate value is computed to match data for the longitude/latitude coordinate reference system, and 1 for planar coordinate reference systems
<code>add</code>	logical. If <code>TRUE</code> add values to current plot
<code>xlim</code>	numeric. min and max values to set the extent of the x-axis
<code>ylim</code>	numeric. min and max values to set the extent of the y-axis
<code>...</code>	graphical parameters as in plot or rasterImage

See Also

[plot](#), [colorize](#), [RGB](#)

Examples

```
b <- rast(system.file("ex/logo.tif", package="terra"))
plotRGB(b)
plotRGB(b, mar=c(2,2,2,2))
plotRGB(b, 3, 2, 1)

b[1000:2000] <- NA
plotRGB(b, 3, 2, 1, stretch='hist')
```

<code>plot_extent</code>	<i>Plot a SpatExtent</i>
--------------------------	--------------------------

Description

Plot a `SpatExtent`. Use [lines](#) to add a `SpatExtent` to an existing map.
See [plot](#) for plotting other object types.

Usage

```
## S4 method for signature 'SpatExtent,missing'
plot(x, y, ...)
```


Arguments

x	SpatExtent
y	missing
...	additional graphical arguments for lines

See Also

[plot](#)

Examples

```
r <- rast()
plot(ext(r))
```

plot_graticule	<i>Plot a graticule</i>
----------------	-------------------------

Description

Plot a SpatGraticule. You can create a SpatGraticule with [graticule](#).

Usage

```
## S4 method for signature 'SpatGraticule,missing'
plot(x, y, background=NULL, col="black", mar=NULL, labels=TRUE,
     retro=FALSE, lab.loc=c(1,1), lab.lon=NULL, lab.lat=NULL, lab.cex=0.65,
     lab.col="black", off.lat=0.25, off.lon=0.25, box=FALSE, box.col="black",
     add=FALSE, ...)
```

Arguments

x	SpatRaster or SpatVector
y	missing or positive integer or name indicating the layer(s) to be plotted
background	background color. If NULL, no background is drawn
mar	numeric vector of length 4 to set the margins of the plot. To make space for the legend you may use something like c(3.1, 3.1, 2.1, 7.1). To fill the plotting canvas, you can use c(0,0,0,0). Use NA to not set the margins
col	character. Color for the graticule lines
labels	logical. If TRUE, show graticule labels
retro	logical. If TRUE, show "retro" instead of decimal labels with the graticule
lab.loc	numeric. The first number indicates where the longitude graticule labels should be drawn (1=bottom, 2=top, NA=not drawn, any other number=top and bottom). The second number indicates where the latitude graticule labels should be drawn (1=left, 2=right, NA=not drawn, any other number=left and right)

lab.lon	positive integers between 1 and the number of labels, indicating which longitude graticule labels should be included
lab.lat	positive integers between 1 and the number of labels, indicating which latitude graticule labels should be included
lab.cex	double. size of the label font
lab.col	character. color of the labels
off.lon	numeric. longitude labels offset
off.lat	numeric. latitude labels offset
box	logical. If TRUE, the outer lines of the graticule are drawn on top with a solid line lty=1
box.col	character. color of the outer lines of the graticule if box=TRUE
add	logical. Add the graticule to the current plot?
...	additional graphical arguments passed to lines

See Also

[graticule](#), [plot](#), [points](#), [lines](#), [polys](#), [image](#), [scatterplot](#), scale bar: [sbar](#), north arrow: [north](#)

Examples

```
g <- graticule(60, 30, crs="+proj=robin")

plot(g, background="azure", col="red", lty=2, box=TRUE)
plot(g, background="azure", col="light gray", lab.loc=c(1,2),
lab.lon=c(2,4,6), lab.lat=3:5, lty=3, retro=TRUE)
```

predict	<i>Spatial model predictions</i>
---------	----------------------------------

Description

Make a SpatRaster object with predictions from a fitted model object (for example, obtained with `glm` or `randomForest`). The first argument is a SpatRaster object with the predictor variables. The [names](#) in the Raster object should exactly match those expected by the model. Any regression like model for which a `predict` method has been implemented (or can be implemented) can be used.

The method should work if the model's `predict` function returns a vector, matrix or data.frame (or a list that can be coerced to a data.frame). In other cases it may be necessary to provide a custom "predict" function that wraps the model's `predict` function to return the values in the required form. See the examples.

This approach of using model predictions is commonly used in remote sensing (for the classification of satellite images) and in ecology, for species distribution modeling.

Usage

```
## S4 method for signature 'SpatRaster'
predict(object, model, fun=predict, ..., const=NULL, na.rm=FALSE,
        index=NULL, cores=1, cpkgs=NULL, filename="", overwrite=FALSE, wopt=list())
```

Arguments

object	SpatRaster
model	fitted model of any class that has a "predict" method (or for which you can supply a similar method as fun argument. E.g. glm, gam, or randomForest
fun	function. The predict function that takes model as first argument. The default value is predict, but can be replaced with e.g. predict.se (depending on the type of model), or your own custom function
...	additional arguments for fun
const	data.frame. Can be used to add a constant value as a predictor variable so that you do not need to make a SpatRaster layer for it
na.rm	logical. If TRUE, cells with NA values in the any of the layers of x are removed from the computation (even if the NA cell is in a layer that is not used as a variable in the model). This option prevents errors with models that cannot handle NA values when making predictions. In most other cases this will not affect the output. However, there are some models that return predicted values even if some (or all) variables are NA
index	integer. To select subset of output variables
cores	positive integer. If cores > 1, a 'parallel' package cluster with that many cores is created and used
cpkgs	character. The package(s) that need to be loaded on the nodes to be able to run the model.predict function (see examples)
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

[interpolate](#) for spatial model prediction

Examples

```
logo <- rast(system.file("ex/logo.tif", package="terra"))
names(logo) <- c("red", "green", "blue")
p <- matrix(c(48, 48, 48, 53, 50, 46, 54, 70, 84, 85, 74, 84, 95, 85,
              66, 42, 26, 4, 19, 17, 7, 14, 26, 29, 39, 45, 51, 56, 46, 38, 31,
              22, 34, 60, 70, 73, 63, 46, 43, 28), ncol=2)
```

```

a <- matrix(c(22, 33, 64, 85, 92, 94, 59, 27, 30, 64, 60, 33, 31, 9,
  99, 67, 15, 5, 4, 30, 8, 37, 42, 27, 19, 69, 60, 73, 3, 5, 21,
  37, 52, 70, 74, 9, 13, 4, 17, 47), ncol=2)

xy <- rbind(cbind(1, p), cbind(0, a))

# extract predictor values for points
e <- extract(logo, xy[,2:3])

# combine with response (excluding the ID column)
v <- data.frame(cbind(pa=xy[,1], e))

#build a model, here with glm
model <- glm(formula=pa~., data=v)

#predict to a raster
r1 <- predict(logo, model)

plot(r1)
points(p, bg='blue', pch=21)
points(a, bg='red', pch=21)

# logistic regression
model <- glm(formula=pa~., data=v, family="binomial")
r1log <- predict(logo, model, type="response")

# to get the probability and standard error
r1se <- predict(logo, model, se.fit=TRUE)

# or provide a custom predict function
predfun <- function(model, data) {
  v <- predict(model, data, se.fit=TRUE)
  cbind(p=as.vector(v$fit), se=as.vector(v$se.fit))
}

r2 <- predict(logo, model, fun=predfun)

# principal components of a SpatRaster
pca <- prcomp(logo)

# or use sampling if cannot process all cell values
sr <- spatSample(logo, 100000, "regular")
pca <- prcomp(sr)

x <- predict(logo, pca)
plot(x)

## parallelization
## Not run:
## simple case with GLM

```

```

model <- glm(formula=pa~., data=v)
p <- predict(logo, model, cores=2)

## The above does not work with a model from a contributed
## package, as the package needs to be loaded in each core.
## Below are three approaches to deal with that

library(randomForest)
rfm <- randomForest(formula=pa~., data=v)

## approach 0 (not parallel)
rp0 <- predict(logo, rfm)

## approach 1, use the "cpkgs" argument
rp1 <- predict(logo, rfm, cores=2, cpkgs="randomForest")

## approach 2, write a custom predict function that loads the package
rfun <- function(mod, dat, ...) {
  library(randomForest)
  predict(mod, dat, ...)
}
rp2 <- predict(logo, rfm, fun=rfun, cores=2)

## approach 3, write a parallelized custom predict function
rfun <- function(mod, dat, ...) {
  ncls <- length(cls)
  nr <- nrow(dat)
  s <- split(dat, rep(1:ncls, each=ceiling(nr/ncls), length.out=nr))
  unlist( parallel::clusterApply(cls, s, function(x, ...) predict(mod, x, ...)) )
}

library(parallel)
cls <- parallel::makeCluster(2)
parallel::clusterExport(cls, c("rfm", "rfun", "randomForest"))
rp3 <- predict(logo, rfm, fun=rfun)
parallel::stopCluster(cls)

plot(c(rp0, rp1, rp2, rp3))

### with two output variables (probabilities for each class)
v$pa <- as.factor(v$pa)
rfm2 <- randomForest(formula=pa~., data=v)
rfp <- predict(logo, rfm2, cores=2, type="prob", cpkgs="randomForest")

## End(Not run)

```

Description

Change the coordinate reference system ("project") of a `SpatVector`, `SpatRaster` or a matrix with coordinates.

Usage

```
## S4 method for signature 'SpatVector'
project(x, y, partial = FALSE)

## S4 method for signature 'SpatRaster'
project(x, y, method, mask=FALSE, align=FALSE,
        gdal=TRUE, res=NULL, origin=NULL, threads=FALSE, filename="", ...)

## S4 method for signature 'SpatExtent'
project(x, from, to)

## S4 method for signature 'matrix'
project(x, from, to)
```

Arguments

x	SpatRaster or SpatVector
y	<p>if (x is a <code>SpatRaster</code>, the preferred approach is for y to be a <code>SpatRaster</code> as well, serving as a template for the geometry (extent and resolution) of the output <code>SpatRaster</code>. Alternatively, you can provide a coordinate reference system (CRS) description.</p> <p>You can use the following formats to define coordinate reference systems: WKT, PROJ.4 (e.g., <code>+proj=longlat +datum=WGS84</code>), or an EPSG code (e.g., <code>"epsg:4326"</code>). But note that the PROJ.4 notation has been deprecated, and you can only use it with the WGS84/NAD83 and NAD27 datums. Other datums are silently ignored.</p> <p>If x is a <code>SpatVector</code>, you can provide a crs definition as discussed above, or any other object from which such a crs can be extracted with crs</p>
partial	logical. If TRUE, geometries that can only partially be represented in the output crs are included in the output
method	<p>character. Method used for estimating the new cell values of a <code>SpatRaster</code>. One of:</p> <p>near: nearest neighbor. This method is fast, and it can be the preferred method if the cell values represent classes. It is not a good choice for continuous values. This is used by default if the first layer of x is categorical.</p> <p>bilinear: bilinear interpolation. This is the default if the first layer of x is numeric (not categorical).</p> <p>cubic: cubic interpolation.</p> <p>cubicspline: cubic spline interpolation.</p> <p>lanczos: Lanczos windowed sinc resampling.</p> <p>sum: the weighted sum of all non-NA contributing grid cells.</p>

	min, q1, med, q3, max, average, mode, rms: the minimum, first quartile, median, third quartile, maximum, mean, mode, or root-mean-square value of all non-NA contributing grid cells.
mask	logical. If TRUE, mask out areas outside the input extent (see example with Robinson projection)
align	logical. If TRUE, and y is a SpatRaster, the template is used for the spatial resolution and origin, but the extent is set such that all of the extent of x is included
gdal	logical. If TRUE the GDAL-warp algorithm is used. Otherwise a slower internal algorithm is used that may be more accurate if there is much variation in the cell sizes of the output raster. Only the near and bilinear algorithms are available for the internal algorithm
res	numeric. Can be used to set the resolution of the output raster if y is a CRS
origin	numeric. Can be used to set the origin of the output raster if y is a CRS
threads	logical. If TRUE multiple threads are used (faster for large files)
filename	character. Output filename
...	additional arguments for writing files as in writeRaster
from	character. Coordinate reference system of x
to	character. Output coordinate reference system

Value

SpatVector or SpatRaster

Note

The PROJ.4 notation of coordinate reference systems has been partly deprecated in the GDAL/PROJ library that is used by this function. You can still use this notation, but **only** with the WGS84 datum. Other datums are silently ignored.

Transforming (projecting) raster data is fundamentally different from transforming vector data. Vector data can be transformed and back-transformed without loss in precision and without changes in the values. This is not the case with raster data. In each transformation the values for the new cells are estimated in some fashion. Therefore, if you need to match raster and vector data for analysis, you should generally transform the vector data.

When using this method with a SpatRaster, the preferable approach is to provide a template SpatRaster as argument y. The template is then another raster dataset that you want your data to align with. If you do not have a template to begin with, you can do `project(x, crs)` and then manipulate the output to get the template you want. For example, where possible use whole numbers for the extent and resolution so that you do not have to worry about small differences in the future. You can use commands like `dim(z) = c(180, 360)` or `res(z) <- 100000`.

The output resolution should generally be similar to the input resolution, but there is no "correct" resolution in raster transformation. It is not obvious what this resolution is if you are using lon/lat data that spans a large North-South extent.

See Also

[crs](#), [resample](#)

Examples

```
## SpatRaster
a <- rast(ncols=40, nrows=40, xmin=-110, xmax=-90, ymin=40, ymax=60,
          crs="+proj=longlat +datum=WGS84")
values(a) <- 1:ncell(a)
newcrs="+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"
b <- rast(ncols=94, nrows=124, xmin=-944881, xmax=935118, ymin=4664377, ymax=7144377, crs=newcrs)
w <- project(a, b)

## SpatVector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
crs(v, proj=TRUE)
cat(crs(v), "\n")

project(v, "+proj=moll")

project(v, "EPSG:2169")
```

quantile

Quantiles of spatial data

Description

Compute quantiles for each cell across the layers of a `SpatRaster`.

You can use `global(x, fun=quantile)` to instead compute quantiles across cells for each layer.

You can also use this method to compute quantiles of the numeric variables of a `SpatVector`.

Usage

```
## S4 method for signature 'SpatRaster'
quantile(x, probs=seq(0, 1, 0.25), na.rm=FALSE, filename="", ...)
```

```
## S4 method for signature 'SpatVector'
quantile(x, probs=seq(0, 1, 0.25), ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>probs</code>	numeric vector of probabilities with values in [0,1]
<code>na.rm</code>	logical. If TRUE, NA's are removed from x before the quantiles are computed
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in <code>writeRaster</code>

Value

SpatRaster with layers representing quantiles

See Also

[app](#)

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
rr <- c(r/2, r, r*2)
qr <- quantile(rr)
qr

## Not run:
# same but slower
qa <- app(rr, quantile)

## End(Not run)

#quantile by layer instead of by cell
qg <- global(r, quantile)
```

query	<i>Query a SpatVectorProxy object</i>
-------	---------------------------------------

Description

Query a SpatVectorProxy to extract a subset

Usage

```
## S4 method for signature 'SpatVectorProxy'
query(x, start=1, n=nrow(x), vars=NULL, where=NULL,
      extent=NULL, filter=NULL)
```

Arguments

x	SpatVectorProxy
start	positive integer. The record to start reading at
n	positive integer. The number of records requested
vars	character. Variable names. Must be a subset of names(x)
where	character. expression like "NAME_1='California' AND ID > 3" , to subset records. Note that start and n are applied after executing the where statement
extent	Spat* object. The extent of the object is used as a spatial filter to select the geometries to read. Ignored if filter is not NULL
filter	SpatVector. Used as a spatial filter to select geometries to read (the convex hull is used for lines or points)

Value

SpatVector

See Also

[vect](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f, proxy=TRUE)
v

x <- query(v, vars=c("ID_2", "NAME_2"), start=5, n=2)
x

query(v, vars=c("ID_2", "NAME_1", "NAME_2"), where="NAME_1='Grevenmacher' AND ID_2 > 6")

## with an extent
e <- ext(5.9, 6.3, 49.9, 50)
x <- query(v, extent=e)

## with polygons
p <- as.polygons(e)
x <- query(v, filter=p)
x
```

rangeFill	<i>Fill layers with a range</i>
-----------	---------------------------------

Description

Fill layers with cell-varying ranges defined by a start and end SpatRaster. The range must start at 1 and end at a user-defined maximum.

For example, for a cell with start=3, end=5 and with limit=8, the output for that cell would be 0,0,1,1,1,0,0,0

Usage

```
## S4 method for signature 'SpatRaster'
rangeFill(x, limit, circular=FALSE, filename="", ...)
```

Arguments

- x SpatRaster with at two layers. The Cell values of the first layer indicate the start of the range (1 based); the cell values are indicate the end of the range
- limit numeric > 1. The range size

circular	logical. If TRUE the values are considered circular, such as the days of the year. In that case, if first > last the layers used are c(first:limit, 1:last). Otherwise, the range would be considered invalid and NA would be used
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also[rapp](#)**Examples**

```
x <- y <- rast(ncol=2, nrow=2)
values(x) <- c(NA, 1:3)
values(y) <- c(NA, 4:6)

r <- rangeFill(c(x, y), 8)
```

rapp*Range-apply*

Description

Apply a function to a range of the layers of a SpatRaster that varies by cell. The range is specified for each cell with one or two SpatRasters (arguments *first* and *last*). For either *first* or *last* you can use a single number instead.

You cannot use single numbers for both *first* and *last* because in that case you could use [app](#) or [Summary-methods](#), perhaps [subsetting](#) the layers of a SpatRaster.

See [selectRange](#) to create a new SpatRaster by extracting one or more values starting at a cell-varying layer.

Usage

```
## S4 method for signature 'SpatRaster'
rapp(x, first, last, fun, ..., allvars=FALSE, fill=NA,
      clamp=FALSE, circular=FALSE, filename="", overwrite=FALSE, wopt=list())
```

Arguments

<code>x</code>	SpatRaster
<code>first</code>	SpatRaster or positive integer between 1 and <code>nlyr(x)</code> , indicating the first layer in the range of layers to be considered
<code>last</code>	SpatRaster or positive integer between 1 and <code>nlyr(x)</code> , indicating the last layer in the range to be considered
<code>fun</code>	function to be applied
<code>...</code>	additional arguments passed to <code>fun</code>
<code>allyrs</code>	logical. If TRUE, values for all layers are passed to <code>fun</code> but the values outside of the range are set to <code>fill</code>
<code>fill</code>	numeric. The fill value for the values outside of the range, for when <code>allyrs=TRUE</code>
<code>clamp</code>	logical. If FALSE and the specified range is outside <code>1:nlyr(x)</code> all cells are considered NA. Otherwise, the invalid part of the range is ignored
<code>circular</code>	logical. If TRUE the values are considered circular, such as the days of the year. In that case, if <code>first > last</code> the layers used are <code>c(first:nlyr(x), 1:last)</code> . Otherwise, the range would be considered invalid and NA would be returned
<code>filename</code>	character. Output filename
<code>overwrite</code>	logical. If TRUE, filename is overwritten
<code>wopt</code>	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

[selectRange](#), [app](#), [Summary-methods](#), [lapp](#), [tapp](#)

Examples

```
r <- rast(ncols=9, nrows=9)
values(r) <- 1:ncell(r)
s <- c(r, r, r, r, r, r)
s <- s * 1:6
s[1:2] <- NA
start <- end <- rast(r)
start[] <- 1:3
end[] <- 4:6
a <- rapp(s, start, end, fun="mean")
b <- rapp(s, start, 2, fun="mean")

# cumsum from start to nlyr(x). return all layers
r <- rapp(s, start, nlyr(s), cumsum, allyrs=TRUE, fill=0)
# return only the final value
rr <- rapp(s, start, nlyr(s), function(i) max(cumsum(i)))
```

rast

Create a *SpatRaster*

Description

Methods to create a *SpatRaster*. These objects can be created from scratch, from a filename, or from another object.

A *SpatRaster* represents a spatially referenced surface divided into three dimensional cells (rows, columns, and layers).

When a *SpatRaster* is created from a file, it does not load the cell (pixel) values into memory (RAM). It only reads the parameters that describe the geometry of the *SpatRaster*, such as the number of rows and columns and the coordinate reference system. The actual values will be read when needed.

Usage

```
## S4 method for signature 'character'
rast(x, subds=0, lyrs=NULL, drivers=NULL, opts=NULL, win=NULL, snap="near", vsi=TRUE)

## S4 method for signature 'missing'
rast(x, nrows=180, ncols=360, nlyrs=1, xmin=-180, xmax=180,
      ymin=-90, ymax=90, crs, extent, resolution, vals, names, time, units)

## S4 method for signature 'SpatRaster'
rast(x, nlyrs=nlyr(x), names, vals, keeptime=TRUE, keepunits=FALSE, props=FALSE)

## S4 method for signature 'matrix'
rast(x, type="", crs="", digits=6, extent=NULL)

## S4 method for signature 'data.frame'
rast(x, type="xyz", crs="", digits=6, extent=NULL)

## S4 method for signature 'array'
rast(x, crs="", extent=NULL)

## S4 method for signature 'list'
rast(x, warn=TRUE)

## S4 method for signature 'SpatRasterDataset'
rast(x)

## S4 method for signature 'SpatVector'
rast(x, ...)

## S4 method for signature 'SpatExtent'
rast(x, ...)
```

Arguments

<code>x</code>	filename (character), missing, <code>SpatRaster</code> , <code>SpatRasterDataset</code> , <code>SpatExtent</code> , <code>SpatVector</code> , matrix, array, list of <code>SpatRaster</code> objects. For other types it will be attempted to create a <code>SpatRaster</code> via <code>(`as(x, "SpatRaster")`)</code>
<code>subds</code>	positive integer or character to select a sub-dataset. If zero or "", all sub-datasets are returned (if possible)
<code>lyrs</code>	positive integer or character to select a subset of layers (a.k.a. "bands")
<code>drivers</code>	character. GDAL drivers to consider
<code>opts</code>	character. GDAL dataset open options
<code>win</code>	<code>SpatExtent</code> to set a window (area of interest)
<code>snap</code>	character. One of "near", "in", or "out", to indicate how the extent of window should be "snapped" to <code>x</code>
<code>vsi</code>	logical. If TRUE, <code>"\\vsicurl\"</code> is prepended to filenames that start with "http"
<code>nrows</code>	positive integer. Number of rows
<code>ncols</code>	positive integer. Number of columns
<code>nlyrs</code>	positive integer. Number of layers
<code>xmin</code>	minimum x coordinate (left border)
<code>xmax</code>	maximum x coordinate (right border)
<code>ymin</code>	minimum y coordinate (bottom border)
<code>ymax</code>	maximum y coordinate (top border)
<code>crs</code>	character. Description of the Coordinate Reference System (map projection) in PROJ.4, WKT or <code>authority:code</code> notation. See crs . If this argument is missing, and the x coordinates are within -360 .. 360 and the y coordinates are within -90 .. 90, longitude/latitude is assigned
<code>keeptime</code>	logical. If FALSE the time stamps are discarded
<code>keepunits</code>	logical. If FALSE the layer units are discarded
<code>props</code>	logical. If TRUE the properties (categories and color-table) are kept
<code>extent</code>	object of class <code>SpatExtent</code> . If present, the arguments <code>xmin</code> , <code>xmax</code> , <code>ymin</code> and <code>ymax</code> are ignored
<code>resolution</code>	numeric vector of length 1 or 2 to set the spatial resolution (see res). If this argument is used, arguments <code>ncols</code> and <code>nrows</code> are ignored
<code>vals</code>	numeric. An optional vector with cell values (if fewer values are provided, these are recycled to reach the number of cells)
<code>names</code>	character. An optional vector with layer names (must match the number of layers)
<code>time</code>	time or date stamps for each layer
<code>units</code>	character. units for each layer
<code>type</code>	character. If the value is not "xyz", the raster has the same number of rows and columns as the matrix. If the value is "xyz", the matrix must have at least two columns, the first with <code>x</code> (or longitude) and the second with <code>y</code> (or latitude) coordinates that represent the centers of raster cells. The additional columns are the values associated with the raster cells.

<code>digits</code>	integer to set the precision for detecting whether points are on a regular grid (a low number of digits is a low precision). Only used when <code>type="xyz"</code>
<code>warn</code>	logical. If TRUE, a warnings about empty rasters may be emitted
<code>...</code>	additional arguments passed on to the <code>rast,missing-method</code>

Details

Files are read with the GDAL library. GDAL guesses the file format from the name, and/or tries reading it with different "drivers" (see [gdal](#)) until it succeeds. In very few cases this may cause a file to be opened with the wrong driver, and some information may be lost. For example, when a netCDF file is opened with the HDF5 driver. You can avoid that by using argument `rast("filename.ncdf", drivers="NETCDF")`

These classes hold a C++ pointer to the data "reference class" and that creates some limitations. They cannot be recovered from a saved R session either or directly passed to nodes on a computer cluster. Generally, you should use [writeRaster](#) to save `SpatRaster` objects to disk (and pass a filename or cell values of cluster nodes). Also see [wrap](#).

Value

`SpatRaster`

See Also

[sds](#) to create a `SpatRasterDataset` (4 dimensions) and [vect](#) for vector (points, lines, polygons) data

Examples

```
# Create a SpatRaster from scratch
x <- rast(nrows=108, ncols=21, xmin=0, xmax=10)

# Create a SpatRaster from a file
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)

s <- rast(system.file("ex/logo.tif", package="terra"))

# Create a skeleton with no associated cell values
rast(s)

# from a matrix
m <- matrix(1:25, nrow=5, ncol=5)
rm <- rast(m)

# from a "xyz" data.frame
d <- as.data.frame(rm, xy=TRUE)
head(d)
rast(d, type="xyz")
```

rasterize

Rasterize vector data

Description

Transfer values associated with the geometries of vector data to a raster

Usage

```
## S4 method for signature 'SpatVector,SpatRaster'
rasterize(x, y, field="", fun, ..., background=NA, touches=FALSE, update=FALSE,
cover=FALSE, by=NULL, filename="", overwrite=FALSE, wopt=list())
```

```
## S4 method for signature 'matrix,SpatRaster'
rasterize(x, y, values=1, fun, ..., background=NA, update=FALSE,
by=NULL, filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatVector or a two-column matrix (point coordinates)
y	SpatRaster
field	character or numeric. If field is a character, it should a variable name in x. If field is numeric it typically is a single number or a vector of length nrow(x). The values are recycled to nrow(x)
values	typically a numeric vector of length 1 or nrow(x). If the length is below nrow(x) the values will be recycled to nrow(x). Only used when x is a matrix. Can also be a matrix or data.frame
fun	summarizing function for when there are multiple geometries in one cell. For lines and polygons you can only use "min", "max" and "sum" For points you can use any function that returns a single number; for example mean, length (to get a count), min or max
...	additional arguments passed to fun
background	numeric. Value to put in the cells that are not covered by any of the features of x. Default is NA
touches	logical. If TRUE, all cells touched by lines or polygons are affected, not just those on the line render path, or whose center point is within the polygon. If touches=TRUE, add cannot be TRUE
update	logical. If TRUE, the values of the input SpatRaster are updated
cover	logical. If TRUE and the geometry of x is polygons, the fraction of a cell that is covered by the polygons is returned. This is estimated by determining presence/absence of the polygon in at least 100 sub-cells (more of there are very few cells)

by	character or numeric value(s) to split x into multiple groups. There will be a separate layer for each group returned. If x is a SpatVector, by can be a column number or name. If x is a matrix, by should be a vector that identifies group membership for each row in x
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also[rasterizeGeom](#), [rasterizeWin](#), [mask](#)**Examples**

```

r <- rast(xmin=0, ncols=18, nrows=18)

# generate points
set.seed(1)
p <- spatSample(r, 1000, xy=TRUE, replace=TRUE)

# rasterize points as a matrix
x <- rasterize(p, r, fun=sum)
y <- rasterize(p, r, value=1:nrow(p), fun=max)

# rasterize points as a SpatVector
pv <- vect(p)
xv <- rasterize(pv, r, fun=sum)

# Polygons
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
r <- rast(v, ncols=75, nrows=100)
z <- rasterize(v, r, "NAME_2")
plot(z)
lines(v)

```

Description

Rasterization of geometric properties of vector data. You can get the count of the number of geometries in each cell; the area covered by polygons; the length of the lines; or the number of lines that cross the boundary of each cell. See [rasterize](#) for standard rasterization (of attribute values associated with geometries).

The area of polygons is intended for summing the area of polygons that are relatively small relative to the raster cells, and for when there may be multiple polygons per cell. See `rasterize(fun="sum")` for counting large polygons and `rasterize(cover=TRUE)` to get the fraction that is covered by larger polygons.

Usage

```
## S4 method for signature 'SpatVector,SpatRaster'
rasterizeGeom(x, y, fun="count", unit="m", filename="", ...)
```

Arguments

x	SpatVector
y	SpatRaster
fun	character. "count", "area", "length", or "crosses"
unit	character. "m" or "km"
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[rasterize](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
r <- rast(v, res=.1)

# length of lines
lms <- as.lines(v)
x <- rasterizeGeom(lms, r, fun="length", "km")

# count of points
set.seed(44)
pts <- spatSample(v, 100)
y <- rasterizeGeom(pts, r)

# area of polygons
```

```
pols <- buffer(pts, 1000)
z <- rasterizeGeom(pols, r, fun="area")
```

rasterizeWin	<i>Rasterize points with a moving window</i>
--------------	--

Description

Rasterize points using a circle (or ellipse) as moving window. For each raster cell, the points (x, y) that fall within the window centered on that cell are considered. A function is used to compute a summary value (e.g. "mean") for the values (z) associated with these points.

This can result in much smoother results compared to the standard [rasterize](#) method.

Usage

```
## S4 method for signature 'SpatVector,SpatRaster'
rasterizeWin(x, y, field, win="circle", pars, fun, ..., cvars=FALSE,
             minPoints=1, fill=NA, filename="", wopt=list())

## S4 method for signature 'data.frame,SpatRaster'
rasterizeWin(x, y, win="circle", pars, fun, ..., cvars=FALSE,
             minPoints=1, fill=NA, filename="", wopt=list())
```

Arguments

x	SpatVector or matrix with at least three columns ((x, y) coordinates and a variable to be rasterized)
y	SpatRaster
field	character. field name in SpatVector x with the values to rasterize
win	character to choose the window type. Can be "circle", "ellipse", "rectangle", or "buffer"
pars	parameters to define the window. If win="circle" or win="buffer", a single number to set the radius of the circle or the width of the buffer. If win="ellipse", either two numbers (the x and y-axis) or three numbers the axes and a rotation (in degrees). If win="rectangle", either two (width, height) or three (width, height) and the rotation in degrees. The unit of the radius/width/height/axis parameters is that of the coordinate reference system. That is, if you have a lon/lat crs, there is no conversion of degrees to meters or vice-versa
fun	function to summarize the values for each cell. If cvars=FALSE, functions must take a numeric vector and return (in all cases) one or more numbers. If cvars=TRUE, and multiple variables are used, the function must take a single argument (a data.frame with the names variables). For win="circle" and win="ellipse" there are two additional character values that can be used: "distto" (average distance to the points from the center of the cell) and "distbetween" (average distance between the points inside the window)

...	additional named arguments passed to fun
minPoints	numeric. The minimum number of points to use. If fewer points are found in a search ellipse it is considered empty and the fill value is returned
fill	numeric. value to use to fill cells with empty search areas
cvars	logical. When using multiple fields, should fun operate on all of them at once? If not, fun is applied to each variable separately
filename	character. Output filename
wopt	list with additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[rasterize](#), [rasterizeGeom](#), [interpNear](#), [interpIDW](#)

Examples

```
r <- rast(ncol=100, nrow=100, crs="local", xmin=0, xmax=50, ymin=0, ymax=50)
set.seed(100)
x <- runif(50, 5, 45)
y <- runif(50, 5, 45)
z <- sample(50)
xyz <- data.frame(x,y,z)

r <- rasterizeWin(xyz, r, fun="count", pars=5)

rfuns <- c("count", "min", "max", "mean")
x <- lapply(rfun, function(f) rasterizeWin(xyz, r, fun=f, pars=5))
names(x) <- rfuns
x <- rast(x)
#plot(x)
```

rcl

Combine row, column, and layer numbers

Description

Get a matrix with the combination of row, column, and layer numbers

Usage

```
## S4 method for signature 'SpatRaster'
rcl(x, row=NULL, col=NULL, lyr=NULL)
```

Arguments

x	SpatRaster
row	positive integer that are row number(s), a list thereof, or NULL for all rows
col	as above for columns
lyr	as above for layers

Details

If a list is used for at least one of row, col or lyr, these are evaluated in parallel. That is combinations are made for each list element, not across list elements. If, in this case another argument is not a list it has to have either length 1 (used for all cases) or have the same length as the (longest) list, in which case the value is coerced into a list with `as.list`

If multiple arguments are a list but they have different lengths, they are recycled to the longest list.

Value

matrix

See Also

[rowColCombine](#), [cellFromRowCol](#)

Examples

```
x <- rast(ncol=5, nrow=5, nlyr=2)
values(x) <- 1:size(x)

rcl(x, 1, 2:3, 1:2)

i <- rcl(x, 1, list(1:2, 3:4), 1:2)
i

# get the values for these cells
x[i]
```

readwrite	<i>Read from, or write to, file</i>
-----------	-------------------------------------

Description

Methods to read from or write chunks of values to or from a file. These are low level methods for programmers. Use `writeRaster` if you want to save an entire `SpatRaster` to file in one step. It is much easier to use.

To write chunks, begin by opening a file with `writeStart`, then write values to it in chunks using the list that is returned by `writeStart`. When writing is done, close the file with `writeStop`.

`blocks` only returns chunk size information. This can be useful when reading, but not writing, raster data.

Usage

```
## S4 method for signature 'SpatRaster'
readStart(x)

## S4 method for signature 'SpatRaster'
readStop(x)

## S4 method for signature 'SpatRaster'
readValues(x, row=1, nrows=nrow(x), col=1, ncol=ncol(x), mat=FALSE, dataframe=FALSE, ...)

## S4 method for signature 'SpatRaster,character'
writeStart(x, filename="", overwrite=FALSE, n=4, sources="", ...)

## S4 method for signature 'SpatRaster'
writeStop(x)

## S4 method for signature 'SpatRaster,vector'
writeValues(x, v, start, nrows)

## S4 method for signature 'SpatRaster'
blocks(x, n=4)

fileBlocksize(x)
```

Arguments

x	SpatRaster
filename	character. Output filename
v	vector with cell values to be written
start	integer. Row number (counting starts at 1) from where to start writing v
row	positive integer. Row number to start from, should be between 1 and nrow(x)
nrows	positive integer. How many rows?
col	positive integer. Column number to start from, should be between 1 and ncol(x)
ncols	positive integer. How many columns? Default is the number of columns left after the start column
mat	logical. If TRUE, values are returned as a numeric matrix instead of as a vector, except when dataframe=TRUE. If any of the layers of x is a factor, the level index is returned, not the label. Use dataframe=TRUE to get the labels
dataframe	logical. If TRUE, values are returned as a data.frame instead of as a vector (also if matrix is TRUE)
overwrite	logical. If TRUE, filename is overwritten
n	positive integer indicating how many copies the data may be in memory at any point in time. This is used to determine how many blocks (large) datasets need to be read

sources	character. Filenames that may not be overwritten because they are used as input to the function. Can be obtained with <code>sources(x)</code>
...	For <code>writeStart</code> : additional arguments for writing files as in writeRaster For <code>readValues</code> : additional arguments for data.frame (and thus only relevant when <code>dataframe=TRUE</code>)

Value

`readValues` returns a vector, matrix, or `data.frame`

`writeStart` returns a list that can be used for processing the file in chunks.

The other methods invisibly return a logical value indicating whether they were successful or not. Their purpose is the side-effect of opening or closing files.

rectify	<i>Rectify a SpatRaster</i>
---------	-----------------------------

Description

Rectify a rotated `SpatRaster` into a non-rotated object

Usage

```
## S4 method for signature 'SpatRaster'
rectify(x, method="bilinear", aoi=NULL, snap=TRUE,
        filename="", ...)
```

Arguments

x	<code>SpatRaster</code> to be rectified
method	character. Method used to for resampling. See resample
aoi	<code>SpatExtent</code> or <code>SpatRaster</code> to crop x to a smaller area of interest; Using a <code>SpatRaster</code> allowing to set the exact output extent and output resolution
snap	logical. If <code>TRUE</code> , the origin and resolution of the output are the same as would the case when <code>aoi = NULL</code> . Only relevant if <code>aoi</code> is a <code>SpatExtent</code>
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

`SpatRaster`

relate	<i>Spatial relationships between geometries</i>
--------	---

Description

`relate` returns a logical matrix indicating the presence or absence of a specific spatial relationships between the geometries in `x` and `y`.

`is.related` returns a logical vector indicating the presence or absence of a specific spatial relationships between `x` and any of the geometries in `y`

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
relate(x, y, relation, pairs=FALSE, na.rm=TRUE)

## S4 method for signature 'SpatVector,missing'
relate(x, y, relation, pairs=FALSE, na.rm=TRUE)

## S4 method for signature 'SpatVector,SpatVector'
is.related(x, y, relation)
```

Arguments

<code>x</code>	SpatVector or SpatExtent
<code>y</code>	missing or as for <code>x</code>
<code>relation</code>	character. One of "intersects", "touches", "crosses", "overlaps", "within", "contains", "covers", "coveredby", "disjoint". Or a "DE-9IM" string such as "FF*FF****". See wikipedia or geotools doc
<code>pairs</code>	logical. If TRUE a two-column matrix is returned with the indices of the cases where the requested relation is TRUE. This is especially helpful when dealing with many geometries as the returned value is generally much smaller
<code>na.rm</code>	logical. If TRUE and <code>sparse=TRUE</code> , geometries in <code>x</code> for which there is no related geometry in <code>y</code> are omitted

Value

matrix (`relate`) or vector (`is.related`)

See Also

[adjacent](#), [nearby](#), [intersect](#), [crop](#)

Examples

```

# polygons
p1 <- vect("POLYGON ((0 0, 8 0, 8 9, 0 9, 0 0))")
p2 <- vect("POLYGON ((5 6, 15 6, 15 15, 5 15, 5 6))")
p3 <- vect("POLYGON ((8 2, 9 2, 9 3, 8 3, 8 2))")
p4 <- vect("POLYGON ((2 6, 3 6, 3 8, 2 8, 2 6))")
p5 <- vect("POLYGON ((2 12, 3 12, 3 13, 2 13, 2 12))")
p6 <- vect("POLYGON ((10 4, 12 4, 12 7, 11 7, 11 6, 10 6, 10 4))")

p <- rbind(p1, p2, p3, p4, p5, p6)
plot(p, col=rainbow(6, alpha=.5))
lines(p, lwd=2)
text(p)

## relate SpatVectors
relate(p1, p2, "intersects")
relate(p1, p3, "touches")
relate(p1, p5, "disjoint")
relate(rbind(p1, p2), p4, "disjoint")

## relate geometries within SpatVectors
# which are completely separated?
relate(p, relation="disjoint")

# which touch (not overlap or within)?
relate(p, relation="touches")
# which overlap (not merely touch, and not within)?
relate(p, relation="overlaps")
# which are within (not merely overlap)?
relate(p, relation="within")

# do they touch or overlap or are within?
relate(p, relation="intersects")

all(relate(p, relation="intersects") ==
     (relate(p, relation="overlaps") |
      relate(p, relation="touches") |
      relate(p, relation="within")))

#for polygons, "coveredby" is "within"
relate(p, relation="coveredby")

# polygons, lines, and points

pp <- rbind(p1, p2)
L1 <- vect("LINESTRING(1 11, 4 6, 10 6)")
L2 <- vect("LINESTRING(8 14, 12 10)")
L3 <- vect("LINESTRING(1 8, 12 14)")
lns <- rbind(L1, L2, L3)
pts <- vect(cbind(c(7,10,10), c(3,5,6)))

```

```

plot(pp, col=rainbow(2, alpha=.5))
text(pp, paste0("POL", 1:2), halo=TRUE)
lines(pp, lwd=2)
lines(lns, col=rainbow(3), lwd=4)
text(lns, paste0("L", 1:3), halo=TRUE)
points(pts, cex=1.5)
text(pts, paste0("PT", 1:3), halo=TRUE, pos=4)

relate(lns, relation="crosses")
relate(lns, pp, relation="crosses")
relate(lns, pp, relation="touches")
relate(lns, pp, relation="intersects")

relate(lns, pp, relation="within")
# polygons can contain lines or points, not the other way around
relate(lns, pp, relation="contains")
relate(pp, lns, relation="contains")
# points and lines can be covered by polygons
relate(lns, pp, relation="coveredby")

relate(pts, pp, "within")
relate(pts, pp, "touches")
relate(pts, lns, "touches")

```

rep

Replicate layers

Description

Replicate layers in a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
rep(x, ...)
```

Arguments

x	SpatRaster
...	arguments as in rep

Value

SpatRaster

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- rep(s, 2)
nlyr(x)
names(x)
x
```

replace_dollar	<i>Replace with \$<-</i>
----------------	-----------------------------

Description

Replace a layer of a `SpatRaster`, or an attribute variable of a `SpatVector`

Usage

```
## S4 replacement method for signature 'SpatRaster'
x$name <- value

## S4 replacement method for signature 'SpatVector'
x$name<-value

## S4 replacement method for signature 'SpatExtent'
x$name <- value
```

Arguments

x	<code>SpatRaster</code> , <code>SpatVector</code> or <code>SpatExtent</code>
name	character. If x is a <code>SpatRaster</code> : layer name. If x is a <code>SpatVector</code> : variable name. If x is a <code>SpatExtent</code> : "xmin", "xmax". "ymin" or "ymax"
value	if x is a <code>SpatRaster</code> , a <code>SpatRaster</code> for which this TRUE: <code>nlyr(value) == length(i)</code> ; if x is a <code>SpatVector</code> , a vector of new values; if x is a <code>SpatExtent</code> a single number

Value

Same as x

See Also

[\[\[<-](#), [\[<-](#), [\\$](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
v$ID_1 <- LETTERS[1:12]
v$new <- sample(12)
values(v)
```

replace_layers	<i>Replace layers or variables</i>
----------------	------------------------------------

Description

Replace the layers of `SpatRaster` with (layers from) another `SpatRaster` or replace variables of a `SpatVector`. You can also create new layers/variables with these methods.

Usage

```
## S4 replacement method for signature 'SpatRaster,numeric'
x[[i]] <- value

## S4 replacement method for signature 'SpatRaster,character'
x[[i]] <- value

## S4 replacement method for signature 'SpatVector,numeric'
x[[i]] <- value

## S4 replacement method for signature 'SpatVector,character'
x[[i]] <- value
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>i</code>	if <code>x</code> is a <code>SpatRaster</code> : layer number(s) of name(s). If <code>x</code> is a <code>SpatVector</code> : variable number(s) or name(s) (column of the attributes)
<code>value</code>	if <code>x</code> is a <code>SpatRaster</code> : <code>SpatRaster</code> for which this TRUE: <code>nlyr(value) == length(i)</code> . if <code>x</code> is a <code>SpatVector</code> : vector or <code>data.frame</code>

Value

`SpatRaster`

See Also

[\\$<-](#), [\[<-](#)

Examples

```
# raster
s <- rast(system.file("ex/logo.tif", package="terra"))
s[["red"]] <- mean(s)
s[[2]] <- sqrt(s[[1]])

# vector
v <- vect(system.file("ex/lux.shp", package="terra"))
v[["ID_1"]] <- 12:1
```

replace_values	<i>Replace values of a SpatRaster</i>
----------------	---------------------------------------

Description

Replace values of a SpatRaster. These are convenience functions for smaller objects only. For larger rasters see [link{classify}](#) or [subst](#)

Usage

```
## S4 replacement method for signature 'SpatRaster,ANY,ANY,ANY'
x[i, j, k] <- value

## S4 replacement method for signature 'SpatVector,ANY,ANY'
x[i, j] <- value

## S4 replacement method for signature 'SpatExtent,numeric,missing'
x[i, j] <- value
```

Arguments

x	SpatRaster
i	row numbers. numeric, logical, or missing for all rows. Can also be a SpatRaster or SpatVector
j	column numbers. numeric, logical or missing for all columns
k	layer number. numeric, logical or missing for all layers
value	numeric, matrix, or data.frame

Value

SpatRaster

See Also

[classify](#), [subst](#), [set.values](#), [values](#), [\[\[<-](#)

Examples

```
## SpatRaster
r <- rast(ncols=5, nrows=5, xmin=0, xmax=5, ymin=0, ymax=5)
r[] <- 1:25
r[1,] <- 5
r[,2] <- 10
r[r>10] <- NA

## SpatVector
f <- system.file("ex/lux.shp", package="terra")
```

```
v <- vect(f)
v[2,2] <- "hello"
v[1,] <- v[10,]
v[,3] <- v[,1]
v[2, "NAME_2"] <- "terra"
head(v, 3)
```

resample	<i>Transfer values of a SpatRaster to another one with a different geometry</i>
----------	---

Description

resample transfers values between SpatRaster objects that do not align (have a different origin and/or resolution). See [project](#) to change the coordinate reference system (crs).

If the origin and extent of the input and output are the same, you should consider using these other functions instead: [aggregate](#), [disagg](#), [extend](#) or [crop](#).

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
resample(x, y, method, threads=FALSE, filename="", ...)
```

Arguments

x	SpatRaster to be resampled
y	SpatRaster with the geometry that x should be resampled to
method	character. Method used for estimating the new cell values. One of: near: nearest neighbor. This method is fast, and it can be the preferred method if the cell values represent classes. It is not a good choice for continuous values. This is used by default if the first layer of x is categorical. bilinear: bilinear interpolation. This is the default if the first layer of x is numeric (not categorical). cubic: cubic interpolation. cubicspline: cubic spline interpolation. lanczos: Lanczos windowed sinc resampling. sum: the weighted sum of all non-NA contributing grid cells. min, q1, med, q3, max, average, mode, rms: the minimum, first quartile, median, third quartile, maximum, mean, mode, or root-mean-square value of all non-NA contributing grid cells.
threads	logical. If TRUE multiple threads are used (faster for large files)
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[aggregate](#), [disagg](#), [crop](#), [project](#),

Examples

```
r <- rast(nrows=3, ncols=3, xmin=0, xmax=10, ymin=0, ymax=10)
values(r) <- 1:ncell(r)
s <- rast(nrows=25, ncols=30, xmin=1, xmax=11, ymin=-1, ymax=11)
x <- resample(r, s, method="bilinear")

opar <- par(no.readonly =TRUE)
par(mfrow=c(1,2))
plot(r)
plot(x)
par(opar)
```

rescale	<i>rescale</i>
---------	----------------

Description

Rescale a SpatVector or SpatRaster. This may be useful to make small [inset](#) maps or for georeferencing.

Usage

```
## S4 method for signature 'SpatRaster'
rescale(x, fx=0.5, fy=fx, x0, y0)

## S4 method for signature 'SpatVector'
rescale(x, fx=0.5, fy=fx, x0, y0)
```

Arguments

- x SpatVector or SpatRaster
- fx numeric > 0. The horizontal scaling factor
- fy numeric > 0. The vertical scaling factor
- x0 numeric. x-coordinate of the center of rescaling. If missing, the center of the extent of x is used
- y0 numeric. y-coordinate of the center of rescaling. If missing, the center of the extent of x is used

Value

Same as x

See Also

[t](#), [shift](#), [flip](#), [rotate](#), [inset](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- rescale(v, 0.2)
plot(v)
lines(w, col="red")
```

RGB	<i>Layers representing colors</i>
-----	-----------------------------------

Description

With RGB you can get or set the layers to be used as Red, Green and Blue when plotting a SpatRaster. Currently, a benefit of this is that [plot](#) will send the object to [plotRGB](#). You can also associated the layers with another color space (HSV, HSI or HSL)

With colorize you can convert a three-layer RGB SpatRaster into other color spaces. You can also convert it into a single-layer SpatRaster with a color-table.

Usage

```
## S4 method for signature 'SpatRaster'
RGB(x)

## S4 replacement method for signature 'SpatRaster'
RGB(x, type="rgb")<-value

## S4 method for signature 'SpatRaster'
colorize(x, to="hsv", alpha=FALSE, stretch=NULL,
grays=FALSE, NAzero=FALSE, filename="", overwrite=FALSE, ...)

## S4 method for signature 'SpatRaster'
has.RGB(x, strict=TRUE)
```

Arguments

x	SpatRaster
value	vector of three (or four) positive integers indicating the layers that are red, green and blue (and optionally a fourth transparency layer). Or NULL to remove the RGB settings

type	character. The color space. One of "rgb" "hsv", "hsi" and "hsl"
to	character. The color space to transform the values to. If x has RGB set, you can transform these to "hsv", "hsi" and "hsl", or use "col" to create a single layer with a color table. You can also use "rgb" to back transform to RGB
alpha	logical. Should an alpha (transparency) channel be included? Only used if x has a color-table and to="rgb"
stretch	character. Option to stretch the values to increase contrast: "lin" (linear) or "hist" (histogram). Only used for transforming RGB to col
grays	logical. If TRUE, a gray-scale color-table is created. Only used for transforming RGB to col
NAzero	logical. If TRUE, NAs are treated as zeros such that a color can be returned if at least one of the three channels has a value. Only used for transforming RGB to ("col")
strict	logical. If TRUE, the function returns FALSE if a color space such as "hsv", "hsi" and "hsl" is used
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
...	additional arguments for writing files as in writeRaster

Examples

```

r <- rast(system.file("ex/logo.tif", package="terra"))
plot(r)
has.RGB(r)
RGB(r) <- NULL
has.RGB(r)
plot(r)
RGB(r) <- c(3,1,2)
plot(r)

RGB(r) <- 1:3
x <- colorize(r, "col")
y <- colorize(r, "hsv")
z <- colorize(y, "rgb")

```

roll

Rolling (moving) functions

Description

Compute "rolling" or "moving" functions, such as the "rolling average"

Usage

```
## S4 method for signature 'SpatRaster'
roll(x, n, fun=mean, type="around", circular=FALSE, na.rm=FALSE, filename="", ...)

## S4 method for signature 'numeric'
roll(x, n, fun=mean, type="around", circular=FALSE, na.rm=FALSE)
```

Arguments

x	SpatRaster or numeric
n	integer > 1. The size of the "window", that is, the number of sequential cells to use in fun
fun	a function like mean, min, max, sum
type	character. One of "around", "to", or "from". The choice indicates which values should be used in the computation. The focal cell is always used. If type is "around", (n-1)/2 before and after the focal cell are also included. If type = "from", n-1 cells are after the focal cell are included. If type = "to", n-1 cells before the focal cell are included. For example, when using n=3 for element 5 of a vector; "around" used elements 4,5,6; "to" used elements 3,4,5, and "from" uses elements 5,6,7
circular	logical. If TRUE, the data are considered to have a circular nature (e.g. days or months of the year), such that there are no missing values before first or after the last value.
na.rm	logical. If TRUE, NA values should be ignored (by fun)
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

Same as x

See Also

[cumsum](#)

Examples

```
## numeric
roll(1:12, 3, mean)
roll(1:12, 3, mean, "to")
roll(1:12, 3, mean, circular=TRUE)

## SpatVector
r <- rast(ncol=2, nrow=2, nlyr=10, vals=1)
r[1,2] = 2
r[2,2] = 4

roll(r, n=3, "sum", "from", na.rm=FALSE) |> values()
```

```
roll(r, n=3, "sum", "from", na.rm=TRUE) |> values()
roll(r, n=3, "sum", "from", circular=TRUE) |> values()

roll(r, n=3, "sum", "to", na.rm=TRUE) |> values()

roll(r, n=3, "sum", "around", circular=TRUE) |> values()
```

rotate	<i>Rotate data along longitude</i>
--------	------------------------------------

Description

Rotate a `SpatRaster` that has longitude coordinates from 0 to 360, to standard coordinates between -180 and 180 degrees (or vice-versa). Longitude between 0 and 360 is frequently used in global climate models.

Rotate a `SpatVector` as for a `SpatRaster` `split=TRUE`, or to correct for coordinates that are connected across the date line (and end up at the "other side" of the longitude scale) are reconnected.

Usage

```
## S4 method for signature 'SpatRaster'
rotate(x, left=TRUE, filename="", ...)

## S4 method for signature 'SpatVector'
rotate(x, longitude=0, split=FALSE, left=TRUE, normalize=FALSE)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>left</code>	logical. If <code>TRUE</code> , rotate to the left, else to the right
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster
<code>longitude</code>	numeric. The longitude around which to rotate
<code>split</code>	logical. Should geometries be split at longitude?
<code>normalize</code>	logical. Should the output be normalized to longitudes between -180 and 180? See normalize.longitude

Value

`SpatRaster`

See Also

[shift](#) and [spin](#)

Examples

```

x <- rast(nrows=9, ncols=18, nl=3, xmin=0, xmax=360)
v <- rep(as.vector(t(matrix(1:ncell(x), nrow=9, ncol=18))), 3)
values(x) <- v
z <- rotate(x)

#SpatVector
p <- rbind(c(3847903, 1983584 ), c(3847903, 5801864), c(8301883, 5801864), c(8301883, 1983584 ))
p <- vect(p, "polygons", crs="+init=EPSG:3347")
d <- densify(p, 100000)
g <- project(d, "+proj=longlat")

x <- rotate(g, 50)
plot(g)
lines(x, col="red")

## rotate countries to 0-360 longitude
#w <- geodata::world(path=".")
#x <- rotate(w, long=0, split=TRUE, left=FALSE)

```

same.crs

Compare coordinate reference systems

Description

The function takes two coordinate reference system descriptions and compares them for equality.

Usage

```
same.crs(x, y)
```

Arguments

x	character, SpatRaster, SpatVector, CRS, or other object that returns something intelligible with <code>crs(x)</code>
y	same types as for x

Value

logical

Examples

```

r <- rast()
same.crs(r, "+proj=longlat")

same.crs(r, "+proj=utm +zone=1")

```

sapp	<i>Apply a terra function that takes only a single layer and returns a SpatRaster to all layers of a SpatRaster</i>
------	---

Description

Apply to all layers of a SpatRaster a function that only takes a single layer SpatRaster and returns a SpatRaster (these are rare). In most cases you can also use `lapply` or `sapply` for this.

Or apply the same method to each sub-dataset (SpatRaster) in a SpatRasterDataset

Usage

```
## S4 method for signature 'SpatRaster'
sapp(x, fun, ..., filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRasterDataset'
sapp(x, fun, ..., filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster or SpatRasterDataset
fun	if x is a SpatRaster: a function that takes a SpatRaster argument and can be applied to each layer of x (e.g. terrain . if x is a SpatRasterDataset: a function that is applied to all layers of the SpatRasters in x (e.g. <code>mean</code>)
...	additional arguments to be passed to fun
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Value

SpatRaster

See Also

[lapp](#), [app](#), [tapp](#), [lapply](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra")) + 1

#SpatRasterDataset
sd <- sds(s*2, s/2)
y <- sapp(sd, mean)
z <- sapp(sd, \(i) 2 * mean(i))
```

sbar

*scale bar***Description**

Add a scale bar to a map

Usage

```
sbar(d, xy=NULL, type="line", divs=2, below="", lonlat=NULL, labels,
adj=c(0.5, -1), lwd=2, xpd=TRUE, ticks=FALSE, scaleby=1, halo=TRUE, ...)
```

Arguments

d	numeric. Distance covered by the scale bar. For the scale bar, it should be in the units of the coordinates of the plot (map), and in km for angular (longitude/latitude) data; see argument lonlat. It can also be missing
xy	numeric. x and y coordinate to place the scale bar. It can also be one of following character values: "bottomleft", "bottom", "bottomright", "topleft", "top", "topright", "left", "right", or NULL
type	for sbar: "line" or "bar"
divs	number of divisions for a bar: 2 or 4
below	character. Text to go below the scale bar (e.g., "kilometers")
lonlat	logical or NULL. If logical, TRUE indicates if the plot is using longitude/latitude coordinates. If NULL this is guessed from the plot's coordinates
labels	vector of three numbers to label the scale bar (beginning, midpoint, end)
adj	adjustment for text placement
lwd	line width for the "line" type of the scale bar
xpd	logical. If TRUE, the scale bar can be outside the plotting area
ticks	logical or numeric. If not FALSE, tick marks are added to a "line" scale bar. The length of the tick marks can be specified
scaleby	numeric. If labels is not provided. The labels are divided by this number. For example, use 1000 to go from m to km
halo	logical. If TRUE the "line" type scale bar gets a white background
...	graphical arguments to be passed to other methods

Value

none

See Also

[north](#), [plot](#), [inset](#)

Examples

```
f <- system.file("ex/meuse.tif", package="terra")
r <- rast(f)
plot(r)
sbar()
sbar(1000, xy=c(178500, 333500), type="bar", divs=4, cex=.8)
sbar(1000, xy="bottomright", divs=3, cex=.8, ticks=TRUE)
north(d=250, c(178550, 332500))

f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r, type="interval")
sbar(20, c(6.2, 50.1), type="bar", cex=.8, divs=4)
sbar(15, c(6.3, 50), type="bar", below="km", label=c(0,7.5,15), cex=.8)
sbar(15, c(6.65, 49.8), cex=.8, label=c(0,"km",15))
north(type=2)
sbar(15, c(6.65, 49.7), cex=.8, label="15 kilometer", lwd=5)
sbar(15, c(6.65, 49.6), divs=4, cex=.8, below="km")
```

scale	<i>Scale values</i>
-------	---------------------

Description

Center and/or scale raster data. For details see [scale](#)

Usage

```
## S4 method for signature 'SpatRaster'
scale(x, center=TRUE, scale=TRUE)
```

Arguments

x	SpatRaster
center	logical or numeric. If TRUE, centering is done by subtracting the layer means (omitting NAs), and if FALSE, no centering is done. If center is a numeric vector (recycled to nlyr(x)), then each layer of x has the corresponding value from center subtracted from it.
scale	logical or numeric. If TRUE, scaling is done by dividing the (centered) layers of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done. If scale is a numeric vector (recycled to nlyr(x)), each layer of x is divided by the corresponding value. Scaling is done after centering.

Value

SpatRaster

See Also

[scale](#)

Examples

```
r <- rast(system.file("ex/logo.tif", package="terra"))
s <- scale(r)

## the equivalent, computed in steps
m <- global(r, "mean")
rr <- r - m[,1]
rms <- global(rr, "rms")
ss <- rr / rms[,1]
```

scatterplot	<i>Scatterplot of two SpatRaster layers</i>
-------------	---

Description

Scatterplot of the values of two SpatRaster layers

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
plot(x, y, maxcell=100000, warn=TRUE, nc, nr,
      maxnl=16, smooth=FALSE, gridded=FALSE, ncol=25, nrow=25, ...)
```

Arguments

x	SpatRaster
y	SpatRaster
maxcell	positive integer. Maximum number of cells to use for the plot
nc	positive integer. Optional. The number of columns to divide the plotting device in (when plotting multiple layers)
nr	positive integer. Optional. The number of rows to divide the plotting device in (when plotting multiple layers)
maxnl	positive integer. Maximum number of layers to plot (for multi-layer objects)
smooth	logical. If TRUE show a smooth scatterplot
gridded	logical. If TRUE the scatterplot is gridded (counts by cells)
warn	boolean. Show a warning if a sample of the pixels is used (for scatterplot only)
ncol	positive integer. Number of columns for gridding
nrow	positive integer. Number of rows for gridding
...	additional graphical arguments

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
plot(s[[1]], s[[2]])
plot(s, sqrt(s[[3:1]]))
```

scoff	<i>Scale (gain) and offset</i>
-------	--------------------------------

Description

These functions can be used to get or set the scale (gain) and offset parameters used to transform values when reading raster data from a file. The parameters are applied to the raw values using the formula below:

$$\text{value} \leftarrow \text{value} * \text{scale} + \text{offset}$$

The default value for scale is 1 and for offset is 0. 'scale' is sometimes referred to as 'gain'.

Note that setting the scale and/or offset are intended to be used with values that are stored in a file. When values are memory, assigning scale or offset values will lead to the immediate computation of new values; in such cases it would be clearer to use [Arith-methods](#).

Usage

```
## S4 method for signature 'SpatRaster'
scoff(x)

## S4 replacement method for signature 'SpatRaster'
scoff(x)<-value
```

Arguments

x	SpatRaster
value	two-column matrix with scale (first column) and offset (second column) for each layer. Or NULL to remove all scale and offset values

Value

matrix or changed SpatRaster

Examples

```
r <- rast(system.file("ex/elev.tif", package="terra"))
minmax(r)
scoff(r)
r[4603]

scoff(r) <- cbind(10, 5)
```

```
minmax(r)
scoff(r)
r[4603]
```

sds

Create a SpatRasterDataset

Description

Methods to create a SpatRasterDataset. This is an object to hold "sub-datasets", each a SpatRaster that in most cases will have multiple layers.

See [describe](#) for getting information about the sub-datasets present in a file.

Usage

```
## S4 method for signature 'missing'
sds(x)

## S4 method for signature 'character'
sds(x, ids=0)

## S4 method for signature 'SpatRaster'
sds(x, ...)

## S4 method for signature 'list'
sds(x)

## S4 method for signature 'array'
sds(x, crs="", extent=NULL)
```

Arguments

x	character (filename), or SpatRaster, or list of SpatRaster objects, or missing. If multiple filenames are provided, it is attempted to make SpatRasters from these, and combine them into a SpatRasterDataset
ids	optional. vector of integer subdataset ids. Ignored if the first value is not a positive integer
crs	character. Description of the Coordinate Reference System (map projection) in PROJ.4, WKT or authority:code notation. If this argument is missing, and the x coordinates are within -360 .. 360 and the y coordinates are within -90 .. 90, longitude/latitude is assigned
extent	SpatExtent
...	additional SpatRaster objects

Value

SpatRasterDataset

See Also

[describe](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- sds(s, s/2)
names(x) <- c("first", "second")
x
length(x)

# extract the second SpatRaster
x[2]

a <- array(1:9, c(3,3,3,3))
sds(a)
```

segregate	<i>segregate</i>
-----------	------------------

Description

Create a SpatRaster with a layer for each class (value, or subset of the values) in the input SpatRaster. For example, if the input has vegetation types, this function will create a layer (presence/absence; dummy variable) for each of these classes.

This is called "one-hot encoding" or "dummy encoding" (for a dummy encoding scheme you can remove (any) one of the output layers as it is redundant).

Usage

```
## S4 method for signature 'SpatRaster'
segregate(x, classes=NULL, keep=FALSE, other=0, round=FALSE, digits=0, filename="", ...)
```

Arguments

x	SpatRaster
classes	numeric. The values (classes) for which layers should be made. If NULL all classes are used
keep	logical. If TRUE, cells that are of the class represented by a layer get that value, rather than a value of 1
other	numeric. Value to assign to cells that are not of the class represented by a layer
round	logical. Should the values be rounded first?
digits	integer. Number of digits to round the values to
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Examples

```
r <- rast(nrows=5, ncols=5)
values(r) <- rep(c(1:4, NA), each=5)
b <- segregate(r)
bb <- segregate(r, keep=TRUE, other=NA)
```

sel

*Spatial selection***Description**

Geometrically subset SpatRaster or SpatVector (to be done) by drawing on a plot (map).

Usage

```
## S4 method for signature 'SpatRaster'
sel(x, ...)

## S4 method for signature 'SpatVector'
sel(x, use="rec", show=TRUE, col="cyan", draw=TRUE, ...)
```

Arguments

x	SpatRaster or SpatVector
use	character indicating what to draw. One of "rec" (rectangle) or "pol" (polygon)
show	logical. If TRUE the selected geometries are shown on the map
col	color to be used for drawing if draw=TRUE
draw	logical. If TRUE the area drawn to select geometries is shown on the map
...	additional graphics arguments for drawing the selected geometries

Value

SpatRaster or SpatVector

See Also

[crop](#) and [intersect](#) to make an intersection and [click](#) and [text](#) to see cell values or geometry attributes.

Use [draw](#) to draw a SpatExtent of SpatVector that you want to keep.

Examples

```
## Not run:
# select a subset of a SpatRaster
r <- rast(nrows=10, ncols=10)
values(r) <- 1:ncell(r)
plot(r)
s <- sel(r) # now click on the map twice

# plot the selection on a new canvas:
x11()
plot(s)

# vector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
plot(v)
x <- sel(v) # now click on the map twice
x

## End(Not run)
```

selectHighest	<i>select cells with high or low values</i>
---------------	---

Description

Identify *n* cells that have the highest of lowest values in the first layer of a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
selectHighest(x, n, low=FALSE)
```

Arguments

<i>x</i>	SpatRaster. Only the first layer is processed
<i>n</i>	The number of cells to select
<i>low</i>	logical. If TRUE, the lowest values are selected instead of the highest values

Value

SpatRaster

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- selectHighest(r, 1000)
y <- selectHighest(r, 1000, TRUE)

m <- merge(y-1, x)
levels(m) <- data.frame(id=0:1, elevation=c("low", "high"))
plot(m)
```

selectRange	<i>Select the values of a range of layers, as specified by cell values in another SpatRaster</i>
-------------	--

Description

Use a single layer SpatRaster to select cell values from different layers in a multi-layer SpatRaster. The values of the SpatRaster to select layers (y) should be whole numbers between 1 and nlyr(x) (values outside this range are ignored).

See [rapp](#) for applying a function to a range of variable size.

See [extract](#) for extraction of values by cell, point, or otherwise.

Usage

```
## S4 method for signature 'SpatRaster'
selectRange(x, y, z=1, repint=0, filename="", ...)
```

Arguments

x	SpatRaster
y	SpatRaster. Cell values must be positive integers. They indicate the first layer to select for each cell
z	positive integer. The number of layers to select
repint	integer > 1 and < nlyr(x) allowing for repeated selection at a fixed interval. For example, if x has 36 layers, and the value of a cell in y=2 and repint = 12, the values for layers 2, 14 and 26 are returned
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[rapp](#), [tapp](#), [extract](#)

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1
s <- c(r, r+2, r+5)
s <- c(s, s)
set.seed(1)
values(r) <- sample(3, ncell(r), replace=TRUE)
x <- selectRange(s, r)

x <- selectRange(s, r, 3)
```

 serialize

serialize and saveRDS for SpatRaster and SpatVector

Description

serialize and saveRDS for SpatRaster and SpatVector. Note that these objects will first be "packed" with [wrap](#), and after unserialize/readRDS they need to be unpacked with [rast](#) or [vect](#).

Use of these functions is not recommended. Especially for SpatRaster it is generally much more efficient to use [writeRaster](#) and [write](#), e.g., a GTiff file.

SpatRaster objects must have all values in memory (that is, the cell values are not in files) to be serialized. These functions use [set.values](#) to load values into memory if needed and if deemed possible given the amount of RAM available.

Usage

```
## S4 method for signature 'SpatRaster'
saveRDS(object, file="", ascii = FALSE, version = NULL, compress=TRUE, refhook = NULL)

## S4 method for signature 'SpatVector'
saveRDS(object, file="", ascii = FALSE, version = NULL, compress=TRUE, refhook = NULL)

## S4 method for signature 'SpatRaster'
serialize(object, connection, ascii = FALSE, xdr = TRUE, version = NULL, refhook = NULL)

## S4 method for signature 'SpatVector'
serialize(object, connection, ascii = FALSE, xdr = TRUE, version = NULL, refhook = NULL)
```

Arguments

object	SpatVector or SpatRaster
file	file name to save object to
connection	see serialize
ascii	see serialize or saveRDS
version	see serialize or saveRDS

compress see [serialize](#) or [saveRDS](#)
refhook see [serialize](#) or [saveRDS](#)
xdr see [serialize](#) or [saveRDS](#)

Value

Packed* object

Examples

```
f <- system.file("ex/lux.shp", package="terra")  
v <- vect(f)  
p <- serialize(v, NULL)  
head(p)  
x <- unserialize(p)  
x
```

setValues	<i>Set the values of raster cells or of geometry attributes</i>
-----------	---

Description

Set cell values of a `SpatRaster` or the attributes of a `SpatVector`. For large `SpatRaster` objects use [init](#) instead to set values.

Usage

```
## S4 replacement method for signature 'SpatRaster,ANY'  
values(x)<-value  
  
## S4 method for signature 'SpatRaster,ANY'  
setValues(x, values, keeptime=TRUE, keepunits=TRUE, keepnames=FALSE, props=FALSE)  
  
## S4 replacement method for signature 'SpatVector,ANY'  
values(x)<-value
```

Arguments

x	<code>SpatRaster</code> or <code>SpatVector</code>
value	For <code>SpatRaster</code> : numeric, matrix or <code>data.frame</code> . The length of the numeric values must match the total number of cells (<code>ncell(x) * nlyr(x)</code>), or be a single value. The number of columns of the matrix or <code>data.frame</code> must match the number of layers of x, and the number of rows must match the number of cells of x. It is also possible to use a matrix with the same number of rows as x and the number of columns that matches <code>ncol(x) * nlyr(x)</code> . For <code>SpatVector</code> : <code>data.frame</code> , matrix, vector, or <code>NULL</code>
values	Same as for value

keeptime	logical. If TRUE the time stamps are kept
keepunits	logical. If FALSE the units are discarded
keepnames	logical. If FALSE the layer names are replaced by the column names in y (if present)
props	logical. If TRUE the properties (categories and color-table) are kept

Value

The same object type as x

See Also

[values](#), [init](#)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- setValues(r, 1:ncell(r))
x
values(x) <- runif(ncell(x))
x
head(x)

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
values(v) <- data.frame(ID=1:12, name=letters[1:12])
head(v)
```

shade

Hill shading

Description

Compute hill shade from slope and aspect layers (both in radians). Slope and aspect can be computed with function [terrain](#).

A hill shade layer is often used as a backdrop on top of which another, semi-transparent, layer is drawn.

Usage

```
shade(slope, aspect, angle=45, direction=0, normalize=FALSE,
      filename="", overwrite=FALSE, ...)
```

Arguments

slope	SpatRaster with slope values (in radians)
aspect	SpatRaster with aspect values (in radians)
angle	The elevation angle(s) of the light source (sun), in degrees
direction	The direction (azimuth) angle(s) of the light source (sun), in degrees
normalize	Logical. If TRUE, values below zero are set to zero and the results are multiplied with 255
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
...	additional arguments for writing files as in writeRaster

References

Horn, B.K.P., 1981. Hill shading and the reflectance map. Proceedings of the IEEE 69(1):14-47

See Also

[terrain](#)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
alt <- disagg(r, 10, method="bilinear")
slope <- terrain(alt, "slope", unit="radians")
aspect <- terrain(alt, "aspect", unit="radians")
hill <- shade(slope, aspect, 40, 270)
plot(hill, col=grey(0:100/100), legend=FALSE, mar=c(2,2,1,4))
plot(alt, col=rainbow(25, alpha=0.35), add=TRUE)

# A better hill shade may be achieved by combining
# different angles and directions. For example

h <- shade(slope, aspect, angle = c(45, 45, 45, 80), direction = c(225, 270, 315, 135))
h <- Reduce(mean, h)
```

sharedPaths	<i>Shared paths</i>
-------------	---------------------

Description

Get shared paths of line or polygon geometries. This can for geometries in a single SpatVector, or between two SpatVectors

Usage

```
## S4 method for signature 'SpatVector'
sharedPaths(x, y=NULL)
```

Arguments

- x SpatVector of lines or polygons
- y missing or SpatVector of lines or polygons

Value

SpatVector

See Also

[gaps](#), [topology](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
plot(v, col="light gray")
text(v, halo=TRUE)

x <- sharedPaths(v)
lines(x, col="red", lwd=2)
text(x, col="blue", halo=TRUE, cex=0.8)
head(x)

z <- sharedPaths(v[3,], v[12,])
```

shift	<i>Shift</i>
-------	--------------

Description

Shift a SpatRaster, SpatVector or SpatExtent to another location.

Usage

```
## S4 method for signature 'SpatRaster'
shift(x, dx=0, dy=0, filename="", ...)

## S4 method for signature 'SpatVector'
shift(x, dx=0, dy=0)

## S4 method for signature 'SpatExtent'
shift(x, dx=0, dy=0)
```

Arguments

x	SpatRaster, SpatVector or SpatExtent
dx	numeric. The shift in horizontal direction
dy	numeric. The shift in vertical direction
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

Same as x

See Also

[flip](#), [rotate](#)

Examples

```
r <- rast(xmin=0, xmax=1, ymin=0, ymax=1)
r <- shift(r, dx=1, dy=-1)

e <- ext(r)
shift(e, 5, 5)
```

sieve	<i>Sieve filter</i>
-------	---------------------

Description

Apply a sieve filter. That is, remove "noise", by changing small clumps of cells with a value that is different from the surrounding cells, to the value of the largest neighbouring clump.

Usage

```
## S4 method for signature 'SpatRaster'
sieve(x, threshold, directions=8, filename="", ...)
```

Arguments

x	SpatRaster, single layer with elevation values. Values should have the same unit as the map units
threshold	positive integer. Only clumps smaller than this threshold will be removed
directions	numeric to indicate which cells are connected. Either 4 to only consider the horizontal and vertical neighbors ("rook"), or 8 to consider the vertical, horizontal and diagonal neighbors
filename	character. Output filename
...	Options for writing files as in writeRaster

See Also

[focal](#)

Examples

```
r <- rast(nrows=18, ncols=18, xmin=0, vals=0, crs="local")
r[2, 5] <- 1
r[5:8, 2:3] <- 2
r[7:12, 10:15] <- 3
r[15:16, 15:18] <- 4
freq(r, bylayer=FALSE)

x <- sieve(r, 8)
y <- sieve(r, 9)
```

simplifyGeom	<i>simplifyGeom geometries</i>
--------------	--------------------------------

Description

Reduce the number of nodes used to represent geometries.

Usage

```
## S4 method for signature 'SpatVector'
simplifyGeom(x, tolerance=0.1, preserveTopology=TRUE, makeValid=TRUE)
```

Arguments

- x SpatVector of lines or polygons
- tolerance numeric. The minimum distance between nodes in units of the crs (i.e. degrees for long/lat)
- preserveTopology logical. If TRUE the topology of output geometries is preserved
- makeValid logical. If TRUE, [makeValid](#) is run after simplification to assure that the output polygons are valid

Value

SpatVector

See Also

[sharedPaths](#), [gaps](#), [is.valid](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- simplifyGeom(v, .02, makeValid=FALSE)
e <- erase(w)
g <- gaps(e)
plot(e, lwd=5, border="light gray")
polys(g, col="red", border="red")
```

sort	<i>Sort a SpatRaster or SpatVector</i>
------	--

Description

Sort the cell values of a SpatRaster across layers. You can also compute the sorting order.

Or sort the records of SpatVector (or data.frame) by specifying the column number(s) or names(s) to sort on.

Usage

```
## S4 method for signature 'SpatRaster'
sort(x, decreasing=FALSE, order=FALSE, filename="", ...)

## S4 method for signature 'SpatVector'
sort(x, v, decreasing=FALSE)
```

Arguments

x	SpatRaster
decreasing	logical. If TRUE, sorting is in decreasing order
order	logical. If TRUE the sorting order is returned instead of the sorted values
filename	character. Output filename
...	additional arguments for writing files as in writeRaster
v	character or numeric indicating the column(s) to sort on

Value

SpatRaster

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
r <- c(r, r/2, r*2)
sort(r)

ord <- sort(r, order=TRUE)
# these two are the same
ord[[1]]
which.min(r)
```

sources	<i>Data sources of a SpatRaster</i>
---------	-------------------------------------

Description

Get the data sources of a SpatRaster or SpatVector or related object. Sources are either files (or similar resources) or "", meaning that they are in memory. You can use hasValues to check if in-memory layers actually have cell values.

Usage

```
## S4 method for signature 'SpatRaster'
sources(x, nlyr=FALSE, bands=FALSE)

## S4 method for signature 'SpatVector'
sources(x)

## S4 method for signature 'SpatRaster'
hasValues(x)

## S4 method for signature 'SpatRaster'
inMemory(x, bylayer=FALSE)
```

Arguments

x	SpatRaster, SpatRasterCollection, SpatVector or SpatVectorProxy
nlyr	logical. If TRUE for each source, the number of layers is returned
bands	logical. If TRUE for each source, the "bands" used, that is, the layer number in the source file, are returned
bylayer	logical. If TRUE a value is returned for each layer instead of for each source

Value

A vector of filenames, or "" when there is no filename, if nlyr and bands are both FALSE. Otherwise a data.frame

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
s <- rast(r)
values(s) <- 1:ncell(s)
rs <- c(r,r,s,r)
sources(rs)
hasValues(r)
x <- rast()
hasValues(x)
```

SpatExtent-class	<i>Class "SpatExtent"</i>
------------------	---------------------------

Description

Objects of class `SpatExtent` are used to define the spatial extent (extremes) of objects of the `SpatRaster` class.

Objects from the Class

You can use the `ext` function to create `SpatExtent` objects, or to extract them from `SpatRaster` objects.

Slots

`ptr`: pointer to the C++ class

Methods

`show` display values of a `SpatExtent` object

Examples

```
e <- ext(-180, 180, -90, 90)
e
```


SpatRaster-class

*SpatRaster class***Description**

A SpatRaster represents a rectangular part of the world that is sub-divided into rectangular cells of equal area (in terms of the units of the coordinate reference system). For each cell can have multiple values ("layers").

An object of the SpatRaster class can point to one or more files on disk that hold the cell values, and/or it can hold these values in memory. These objects can be created with the [rast](#) method.

A SpatRasterDataset is a collection of sub-datasets, where each is a SpatRaster for the same area (extent) and coordinate reference system, but possibly with a different resolution. Sub-datasets are often used to capture variables (e.g. temperature and precipitation), or a fourth dimension (e.g. height, depth or time) if the sub-datasets already have three dimensions (multiple layers).

A SpatRasterCollection is a collection of SpatRasters with no restriction in the extent or other geometric parameters.

Examples

```
rast()
```

spatSample

*Take a regular sample***Description**

Take a spatial sample from a SpatRaster, SpatVector or SpatExtent. Sampling a SpatVector or SpatExtent always returns a SpatVector of points.

With a SpatRaster, you can get cell values, cell numbers (cells=TRUE), coordinates (xy=TRUE) or (when type="regular" and as.raster=TRUE) get a new SpatRaster with the same extent, but fewer cells.

In order to assure regularity when requesting a regular sample, the number of cells or points returned may not be exactly the same as the size requested.

Usage

```
## S4 method for signature 'SpatRaster'
spatSample(x, size, method="random", replace=FALSE, na.rm=FALSE,
  as.raster=FALSE, as.df=TRUE, as.points=FALSE, values=TRUE, cells=FALSE,
  xy=FALSE, ext=NULL, warn=TRUE, weights=NULL, exp=5, exhaustive=FALSE)
```

```
## S4 method for signature 'SpatVector'
spatSample(x, size, method="random", strata=NULL, chess="")
```

```
## S4 method for signature 'SpatExtent'
spatSample(x, size, method="random", lonlat, as.points=FALSE)
```

Arguments

x	SpatRaster, SpatVector or SpatExtent
size	numeric. The sample size. If x is a SpatVector, you can also provide a vector of the same length as x in which case sampling is done separately for each geometry. If x is a SpatRaster, and you are using method="regular" you can specify the size as two numbers (number of rows and columns)
method	character. Should be "regular" or "random", If x is a SpatRaster, it can also be "stratified" (each value in x is a stratum) or "weights" (each value in x is a probability weight)
replace	logical. If TRUE, sampling is with replacement (if method="random")
na.rm	logical. If TRUE, NAs are removed. Only used with random sampling of cell values. That is with method="random", as.raster=FALSE, cells=FALSE
as.raster	logical. If TRUE, a SpatRaster is returned
as.df	logical. If TRUE, a data.frame is returned instead of a matrix
as.points	logical. If TRUE, a SpatVector of points is returned
values	logical. If TRUE cell values are returned
cells	logical. If TRUE, cell numbers are returned. If method="stratified" this is always set to TRUE if xy=FALSE
xy	logical. If TRUE, cell coordinates are returned
ext	SpatExtent or NULL to restrict sampling to a subset of the area of x
warn	logical. Give a warning if the sample size returned is smaller than requested
weights	SpatRaster. Used to provide weights when method="stratified"
strata	if not NULL, stratified random sampling is done, taking size samples from each stratum. If x has polygon geometry, strata must be a field name (or index) in x. If x has point geometry, strata can be a SpatVector of polygons or a SpatRaster
chess	character. One of "", "white", or "black". For stratified sampling if strata is a SpatRaster. If not "", samples are only taken from alternate cells, organized like the "white" or "black" fields on a chessboard
lonlat	logical. If TRUE, sampling of a SpatExtent is weighted by cos(latitude). For SpatRaster and SpatVector this done based on the crs , but it is ignored if as.raster=TRUE
exp	numeric >= 1. "Expansion factor" that is multiplied with size to get an initial sample used for stratified samples and random samples with na.rm=TRUE to try to get at least size samples
exhaustive	logical. If TRUE and na.rm=TRUE first all cells that are not NA are determined and a sample is taken from these cells. This is useful when you are dealing with a very large raster that is sparse (most cells are NA). Otherwise, the default approach may not find enough samples. This should not be used in other cases, especially not with large rasters that mostly have values

Value

numeric matrix, data.frame, SpatRaster or SpatVector

Examples

```

f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
s <- spatSample(r, 10, as.raster=TRUE)
spatSample(r, 5)
spatSample(r, 5, na.rm=TRUE)
spatSample(r, 5, "regular")

## if you require cell numbers and/or coordinates
size <- 6
spatSample(r, 6, "random", cells=TRUE, xy=TRUE, values=FALSE)

# regular, with values
spatSample(r, 6, "regular", cells=TRUE, xy=TRUE)

# stratified
rr <- rast(ncol=10, nrow=10, names="stratum")
set.seed(1)
values(rr) <- round(runif(ncell(rr), 1, 3))
spatSample(rr, 2, "stratified", xy=TRUE)

s <- spatSample(rr, 5, "stratified", as.points=TRUE)
plot(rr, plg=list(title="raster"))
plot(s, 1, add=TRUE, plg=list(x=185, y=1, title="points"))

## SpatExtent
e <- ext(r)
spatSample(e, 10, "random", lonlat=TRUE)

## SpatVector
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)

# sample the geometries
i <- sample(v, 3)

# sample points in geometries
p <- spatSample(v, 3)

```

SpatVector-class

*Class "SpatVector"***Description**

SpatVector can represent points, lines or polygons.

SpatVectorCollection can hold a collection of SpatVectors

SpatVectorProxy is a SpatVector for which the data are on-disk in-stead of in memory.

spin	<i>spin a SpatVector</i>
------	--------------------------

Description

Spin (rotate) the geometry of a SpatVector.

Usage

```
## S4 method for signature 'SpatVector'
spin(x, angle, x0, y0)
```

Arguments

x	SpatVector
angle	numeric. Angle of rotation in degrees
x0	numeric. x-coordinate of the center of rotation. If missing, the center of the extent of x is used
y0	numeric. y-coordinate of the center of rotation. If missing, the center of the extent of x is used

Value

SpatVector

See Also

[rescale](#), [t](#), [shift](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
w <- spin(v, 180)
plot(v)
lines(w, col="red")

# lower-right corner as center
e <- as.vector(ext(v))
x <- spin(v, 45, e[1], e[3])
```

split

*Split***Description**

Split a `SpatRaster` by layer, or a `SpatVector` by attributes. You can also split the geometry of a polygon `SpatVector` with a line `SpatVector`.

Usage

```
## S4 method for signature 'SpatRaster,ANY'
split(x, f)
```

```
## S4 method for signature 'SpatVector,ANY'
split(x, f)
```

```
## S4 method for signature 'SpatVector,SpatVector'
split(x, f)
```

Arguments

`x` `SpatRaster` or `SpatVector`

`f` If `x` is a `SpatRaster`: a vector of the length `nlyr(x)`. If `x` is a `SpatVector`: a field (variable) name or a vector of the same length as `x`; or, if `x` is a `SpatVector` of polygons, a `SpatVector` of lines to split the polygon geometries

Value

list or `SpatVector`

Examples

```
## split layers
s <- rast(system.file("ex/logo.tif", package="terra"))
y <- split(s, c(1,2,1))
sds(y)

## split attributes
v <- vect(system.file("ex/lux.shp", package="terra"))
x <- split(v, "NAME_1")

## split geometries
v <- v[1:5,]
line <- vect(matrix(c(5.79, 6.22, 5.75, 6.1, 5.8,
50.14, 50.05, 49.88, 49.85, 49.71), ncol=2), "line")
s <- split(v, line)
```

sprc

*Create a SpatRasterCollection***Description**

Methods to create a SpatRasterCollection. This is an object to hold a collection (list) of SpatRaster objects. There are no restrictions on the similarity of the SpatRaster geometry.

They can be used to combine several SpatRasters to be used with [merge](#) or [mosaic](#)

You can create a SpatRasterCollection from a file with subdatasets.

Usage

```
## S4 method for signature 'SpatRaster'
sprc(x, ...)

## S4 method for signature 'list'
sprc(x)

## S4 method for signature 'missing'
sprc(x)

## S4 method for signature 'character'
sprc(x, ids=0)
```

Arguments

x	SpatRaster, list with SpatRaster objects, missing, or filename
ids	optional. vector of integer subdataset ids. Ignored if the first value is not a positive integer
...	additional SpatRaster objects

Value

SpatRasterCollection

See Also

[sds](#)

Examples

```
x <- rast(xmin=-110, xmax=-50, ymin=40, ymax=70, ncols=60, nrows=30)
y <- rast(xmin=-80, xmax=-20, ymax=60, ymin=30)
res(y) <- res(x)
values(x) <- 1:ncell(x)
values(y) <- 1:ncell(y)
```

```
z <- sprc(x, y)
z
```

stretch

Stretch

Description

Linear or histogram equalization stretch of values in a `SpatRaster`.

For linear stretch, provide the desired output range (`minv` and `maxv`) and the lower and upper bounds in the original data, either as quantiles (`minq` and `maxq`, or as cell values (`smin` and `smax`). If `smin` and `smax` are both not NA, `minq` and `maxq` are ignored.

For histogram equalization, these arguments are ignored, but you can provide the desired scale of the output.

Usage

```
## S4 method for signature 'SpatRaster'
stretch(x, minv=0, maxv=255, minq=0, maxq=1, smin=NA, smax=NA,
histeq=FALSE, scale=1, filename="", ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>minv</code>	numeric ≥ 0 and smaller than <code>maxv</code> . lower bound of stretched value
<code>maxv</code>	numeric ≤ 255 and larger than <code>minv</code> . upper bound of stretched value
<code>minq</code>	numeric ≥ 0 and smaller than <code>maxq</code> . lower quantile bound of original value. Ignored if <code>smin</code> is supplied
<code>maxq</code>	numeric ≤ 1 and larger than <code>minq</code> . upper quantile bound of original value. Ignored if <code>smax</code> is supplied
<code>smin</code>	numeric $< smax$. user supplied lower value for the layers, to be used instead of a quantile computed by the function itself
<code>smax</code>	numeric $> smin$. user supplied upper value for the layers, to be used instead of a quantile computed by the function itself
<code>histeq</code>	logical. If TRUE histogram equalization is used instead of linear stretch
<code>scale</code>	numeric. The scale (maximum value) of the output if <code>histeq=TRUE</code>
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster

Value

`SpatRaster`

Examples

```
r <- rast(nc=10, nr=10)
values(r) <- rep(1:25, 4)
rs <- stretch(r)
s <- c(r, r*2)
sr <- stretch(s)
```

subset

Subset a SpatRaster or a SpatVector

Description

Select a subset of layers from a SpatRaster or select a subset of records (row) and/or variables (columns) from a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
subset(x, subset, negate=FALSE, NSE=FALSE, filename="", overwrite=FALSE, ...)
```

```
## S4 method for signature 'SpatVector'
subset(x, subset, select, drop=FALSE, NSE=FALSE)
```

Arguments

x	SpatRaster or SpatVector
subset	if x is a SpatRaster: integer or character to select layers if x is a SpatVector: logical expression indicating the rows to keep (missing values are taken as FALSE)
select	expression, indicating columns to select
negate	logical. If TRUE all layers that are not in the subset are selected
NSE	logical. If TRUE, non-standard evaluation (the use of unquoted variable names) is allowed. Set this to FALSE when calling subset from a function
drop	logical. If TRUE, the geometries will be dropped, and a data.frame is returned
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
...	additional arguments for writing files as in writeRaster

Value

if x is a SpatRaster: SpatRaster
if x is a SpatVector: SpatVector or, if drop=TRUE, a data.frame.

Examples

```

### SpatRaster
s <- rast(system.file("ex/logo.tif", package="terra"))
subset(s, 2:3)
subset(s, c(3,2,3,1))

#equivalent to
s[[ c(3,2,3,1) ]]

s[[c("red", "green")]]
s$red

# expression based (partial) matching of names with single brackets
s["re"]
s["^re"]

# not with double brackets
# s[["re"]]

### SpatVector
v <- vect(system.file("ex/lux.shp", package="terra"))

subset(v, v$NAME_1 == "Diekirch", c("NAME_1", "NAME_2"))

subset(v, NAME_1 == "Diekirch", c(NAME_1, NAME_2), NSE=TRUE)

# or like this
v[2:3,]
v[1:2, 2:3]
v[1:2, c("NAME_1", "NAME_2")]

```

subset_dollar

*Subset a SpatRaster or a SpatVector***Description**

Select a subset of layers from a SpatRaster or select a subset of records (row) and/or variables (columns) from a SpatVector.

Usage

```

## S4 method for signature 'SpatExtent'
x$name

```

Arguments

x	SpatRaster, SpatVector or SpatExtent
name	character. If x is a SpatRaster: layer name. If x is a SpatVector: variable name. If x is a SpatExtent: xmin, xmax, ymin or ymax

Value

if x is a SpatRaster: SpatRaster

if x is a SpatVector: SpatVector or, if drop=TRUE, a data.frame.

See Also

link{subset}, link{[]}, link{[[}], link{extract}

Examples

```
### SpatRaster
s <- rast(system.file("ex/logo.tif", package="terra"))
subset(s, 2:3)
subset(s, c(3,2,3,1))
#equivalent to
s[[ c(3,2,3,1) ]]

s[[c("red", "green")]]
s$red

# expression based (partial) matching of names with single brackets
s["re"]
s["^re"]

# not with double brackets
# s[["re"]]

### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))
v[2:3,]
v[1:2, 2:3]

subset(v, v$NAME_1 == "Diekirch", c("NAME_1", "NAME_2"))

subset(v, NAME_1 == "Diekirch", c(NAME_1, NAME_2), NSE=TRUE)
```

subset_double	<i>Subset a SpatRaster or a SpatVector</i>
---------------	--

Description

Select a subset of layers from a `SpatRaster` or select a subset of records (row) and/or variables (columns) from a `SpatVector`.

Usage

```
## S4 method for signature 'SpatRaster,numeric,missing'
x[[i, j]]

## S4 method for signature 'SpatRasterDataset,ANY,ANY'
x[[i, j, drop=TRUE]]

## S4 method for signature 'SpatVector,numeric,missing'
x[[i, j, drop=FALSE]]
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>i</code>	if <code>x</code> is a <code>SpatRaster</code> : integer, logical, or character to select layers if <code>x</code> is a <code>SpatVector</code> : integer, logical, or character to select variables)
<code>j</code>	missing, or, for <code>SpatRasterDataset</code> only, numeric
<code>drop</code>	logical. If <code>TRUE</code> , the geometries will be dropped, and a <code>data.frame</code> is returned

Value

if `x` is a `SpatRaster` or `SpatRasterDataset`: `SpatRaster`
if `x` is a `SpatVector`: a `data.frame`.

See Also

`link{subset}`, `link{}`, `link{[]}`, `link{extract}`

Examples

```
### SpatRaster
s <- rast(system.file("ex/logo.tif", package="terra"))
s[[ 1:2 ]]

s[[c("red", "green")]]

# expression based (partial) matching of names with single brackets
s["re"]
s["^re"]
```

```
# does not work with double brackets
# s[["re"]]

### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))
v[[2:3]]

# to keep the geometry use
v[,2:3]
```

subset_single

*Extract values from a SpatRaster, SpatVector or SpatExtent***Description**

Extract values from a SpatRaster; a subset of records (row) and/or variables (columns) from a SpatVector; or a number from a SpatExtent.

You can use indices (row, column, layer or cell numbers) to extract. You can also use other Spat* objects.

Usage

```
## S4 method for signature 'SpatRaster,ANY,ANY,ANY'
x[i, j, k]

## S4 method for signature 'SpatVector,numeric,numeric'
x[i, j, drop=FALSE]

## S4 method for signature 'SpatVector,SpatVector,missing'
x[i, j]

## S4 method for signature 'SpatExtent,numeric,missing'
x[i, j]
```

Arguments

x	SpatRaster, SpatVector or SpatExtent
i	if x is a SpatRaster: numeric, logical or missing to select rows or, if j is missing, to select cells numbers. if x is a SpatVector: numeric or missing to select rows. if i is another SpatVector: get a new SpatVector with the geometries that intersect. if x is a SpatExtent: integer between 1 and 4.
j	numeric, logical, or missing to select columns

k

numeric, character, or missing to select layers

drop

logical. If FALSE an object of the same class as x is returned

Value

numeric if x is a SpatExtent. Same as x if drop=FALSE. Otherwise a data.frame

See Also

linkextract, link{subset}, link{\$}, link{[[},

Examples

```
### SpatRaster
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
r[3638]
rowColFromCell(r, 2638)
r[39, 28]
x <- r[39:40, 28:29, drop=FALSE]
as.matrix(x, wide=TRUE)

### SpatVector

v <- vect(system.file("ex/lux.shp", package="terra"))
v[2:3,]
v[1:2, 2:3]
v[1:2, 2:3, drop=TRUE]
```

subst	<i>replace cell values</i>
-------	----------------------------

Description

Substitute(replace) cell values of a SpatRaster with a new value. See [classify](#) for more complex/flexible replacement.

Usage

```
## S4 method for signature 'SpatRaster'
subst(x, from, to, others=NULL, raw=FALSE, filename="", ...)
```

Arguments

x

SpatRaster

from

numeric value(s). Normally a vector of the same length as ‘to’. If x has multiple layers, it can also be a matrix of numeric value(s) where nrow(x) == length(to). In that case the output has a single layer, with values based on the combination of the values of the input layers

to	numeric value(s). Normally a vector of the same length as 'from'. If x has a single layer, it can also be a matrix of numeric value(s) where nrow(x) == length(from). In that case the output has multiple layers, one for each column in to
others	numeric. If not NULL all values that are not matched are set to this value. Otherwise they retain their original value.
raw	logical. If TRUE, the values in from and to are the raw cell values, not the categorical labels. Only relevant if is.factor(x)
filename	character. Output filename
...	Additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[classify](#)

Examples

```
r <- rast(ncols=5, nrows=5, xmin=0, xmax=1, ymin=0, ymax=1, crs="")
r <- init(r, 1:6)
x <- subst(r, 3, 7)
x <- subst(r, 2:3, NA)
x <- subst(x, NA, 10)

# multiple output layers
z <- subst(r, 2:3, cbind(20,30))

# multiple input layers
rr <- c(r, r+1, r+2)
m <- rbind(c(1:3), c(3:5))
zz <- subst(rr, m, c(100, 200))
```

summarize	<i>Summarize</i>
-----------	------------------

Description

Compute summary statistics for cells, either across layers or between layers (parallel summary). The following summary methods are available for SpatRaster: any, anyNA, all, allNA, max, min, mean, median, prod, range, stdev, sum, which.min, which.max. See [modal](#) to compute the mode and [app](#) to compute summary statistics that are not included here. Because generic functions are used, the method applied is chosen based on the first argument: "x". This means that if r is a SpatRaster, mean(r, 5) will work, but mean(5, r) will not work.

The mean method has an argument "trim" that is ignored.

If pop=TRUE stdev computes the population standard deviation, computed as:

```
f <- function(x) sqrt(sum((x-mean(x))^2) / length(x))
```

This is different than the sample standard deviation returned by sd (which uses n-1 as denominator).

Usage

```
## S4 method for signature 'SpatRaster'
min(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
max(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
range(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
prod(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
sum(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
any(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
all(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
range(x, ..., na.rm=FALSE)

## S4 method for signature 'SpatRaster'
which.min(x)

## S4 method for signature 'SpatRaster'
which.max(x)

## S4 method for signature 'SpatRaster'
stdev(x, ..., pop=TRUE, na.rm=FALSE)

## S4 method for signature 'SpatRaster'
mean(x, ..., trim=NA, na.rm=FALSE)

## S4 method for signature 'SpatRaster'
median(x, na.rm=FALSE, ...)

## S4 method for signature 'SpatRaster'
anyNA(x)
```

```
## S4 method for signature 'SpatRaster'
countNA(x, n=0)

## S4 method for signature 'SpatRaster'
noNA(x, falseNA=FALSE)

## S4 method for signature 'SpatRaster'
allNA(x, falseNA=FALSE)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>...</code>	additional <code>SpatRaster</code> objects or numeric values; and arguments <code>par</code> for parallel summarization (see Details), and <code>filename</code> , <code>overwrite</code> and <code>wopt</code> as for writeRaster
<code>na.rm</code>	logical. If TRUE, NA values are ignored. If FALSE, NA is returned if <code>x</code> has any NA values
<code>trim</code>	ignored
<code>pop</code>	logical. If TRUE, the population standard deviation is computed. Otherwise the sample standard deviation is computed
<code>falseNA</code>	logical. If TRUE, cells that would otherwise be FALSE are set to NA
<code>n</code>	integer. If <code>n > 0</code> , cell values are TRUE if at least <code>n</code> of its layers are NA

Details

Additional argument `par` can be used for "parallel" summarizing a `SpatRaster` and a numeric or logical value. If a `SpatRaster` `x` has three layers, `max(x, 5)` will return a single layer (the number five is treated as a layer in which all cells have value five). In contrast `max(x, 5, par=TRUE)` returns three layers (the number five is treated as another `SpatRaster` with a single layer with all cells having the value five).

Value

`SpatRaster`

See Also

[app](#), [Math-methods](#), [modal](#), [which.lyr](#)

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10, nlyrs=3)
values(r) <- runif(ncell(r) * nlyr(r))

x <- mean(r)
# note how this returns one layer
```



```

x <- sum(c(r, r[[2]]), 5)

# and this returns three layers
y <- sum(r, r[[2]], 5)

max(r)

## when adding a number, do you want 1 layer or all layers?
# 1 layer
max(r, 0.5)

# all layers
max(r, 0.5, par=TRUE)

y <- stdev(r)
# not the same as
yy <- app(r, sd)

z <- stdev(r, r*2)

x <- mean(r, filename=paste0(tempfile(), ".tif"))

v <- values(r)
set.seed(3)
v[sample(length(v), 50)] <- NA
values(r) <- v
is.na(r)
anyNA(r)
allNA(r)
countNA(r)
countNA(r, 2)

```

summary

summary

Description

Compute summary statistics (min, max, mean, and quartiles) for SpatRaster using base [summary](#) method. A sample is used for very large files.

For single or other statistics see [Summary-methods](#), [global](#), and [quantile](#)

Usage

```

## S4 method for signature 'SpatRaster'
summary(object, size=100000, warn=TRUE, ...)

## S4 method for signature 'SpatVector'
summary(object, ...)

```

Arguments

object	SpatRaster or SpatVector
size	positive integer. Size of a regular sample used for large datasets (see spatSample)
warn	logical. If TRUE a warning is given if a sample is used
...	additional arguments passed on to the base summary method

Value

matrix with (an estimate of) the median, minimum and maximum values, the first and third quartiles, and the number of cells with NA values

See Also

[Summary-methods](#), [global](#), [quantile](#)

Examples

```
set.seed(0)
r <- rast(nrows=10, ncols=10, nlyrs=3)
values(r) <- runif(nlyr(r)*ncell(r))
summary(r)
```

svc	<i>Create a SpatVectorCollection</i>
-----	--------------------------------------

Description

Methods to create a SpatVectorCollection. This is an object to hold "sub-datasets", each a SpatVector, perhaps of different geometry type.

Usage

```
## S4 method for signature 'missing'
svc(x)

## S4 method for signature 'SpatVector'
svc(x, ...)

## S4 method for signature 'list'
svc(x)
```

Arguments

x	SpatVector, or list of a SpatVector, or missing
...	Additional SpatVectors

Value

SpatVectorCollection

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
x <- svc()
x <- svc(v, v[1:3,], as.lines(v[3:5,]), as.points(v))
length(x)
x

# extract
x[3]

# replace
x[2] <- as.lines(v[1,])
```

symdif	<i>Symmetrical difference</i>
--------	-------------------------------

Description

Symmetrical difference of polygons

Usage

```
## S4 method for signature 'SpatVector,SpatVector'
symdif(x, y)
```

Arguments

x	SpatVector
y	SpatVector

Value

SpatVector

See Also

[erase](#)

Examples

```
p <- vect(system.file("ex/lux.shp", package="terra"))
b <- as.polygons(ext(6, 6.4, 49.75, 50))
#sd <- symdif(p, b)
#plot(sd, col=rainbow(12))
```

tapp

Apply a function to subsets of layers of a SpatRaster

Description

Apply a function to subsets of layers of a SpatRaster (similar to [tapply](#) and [aggregate](#)). The layers are combined based on the index.

The number of layers in the output SpatRaster equals the number of unique values in index times the number of values that the supplied function returns for a single vector of numbers.

For example, if you have a SpatRaster with 6 layers, you can use `index=c(1,1,1,2,2,2)` and `fun=sum`. This will return a SpatRaster with two layers. The first layer is the sum of the first three layers in the input SpatRaster, and the second layer is the sum of the last three layers in the input SpatRaster. Indices are recycled such that `index=c(1,2)` would also return a SpatRaster with two layers (one based on the odd layers (1,3,5), the other based on the even layers (2,4,6)).

The index can also be one of the following values to group by time period (if x has the appropriate [time](#) values): "years", "months", "yearmonths", "week" (the ISO 8601 week number, see Details), "yearweeks", "days", "doy" (day of the year), "7days" (seven-day periods starting at Jan 1 of each year), "10days", or "15days". It can also be a function that makes groups from time values.

See [app](#) or [Summary-methods](#) if you want to use a more efficient function that returns multiple layers based on **all** layers in the SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
tapp(x, index, fun, ..., cores=1, filename="", overwrite=FALSE, wopt=list())
```

Arguments

x	SpatRaster
index	factor or numeric (integer). Vector of length <code>nlyr(x)</code> (shorter vectors are recycled) grouping the input layers. It can also be one of the following values: "years", "months", "yearmonths", "days", "week" (ISO 8601 week number), or "doy" (day of the year)
fun	function to be applied. The following functions have been re-implemented in C++ for speed: "sum", "mean", "median", "modal", "which", "which.min", "which.max", "min", "max", "prod", "any", "all", "sd", "std", "first". To use the base-R function for say, "min", you could use something like <code>fun = \((i) min(i)</code>
...	additional arguments passed to fun

cores	positive integer. If cores > 1, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
filename	character. Output filename
overwrite	logical. If TRUE, filename is overwritten
wopt	list with named options for writing files as in writeRaster

Details

"week" follows the ISO 8601 definition. Weeks start on Monday. If the week containing 1 January has four or more days in the new year, then it is considered week "01". Otherwise, it is the last week of the previous year (week "52" or "53", and the next week is week 1.

Value

SpatRaster

See Also

[app](#), [Summary-methods](#)

Examples

```
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
s <- c(r, r, r, r, r, r)
s <- s * 1:6
b1 <- tapp(s, index=c(1,1,1,2,2,2), fun=sum)
b1
b2 <- tapp(s, c(1,2,3,1,2,3), fun=sum)
b2
```

terrain	<i>terrain characteristics</i>
---------	--------------------------------

Description

Compute terrain characteristics from elevation data. The elevation values should be in the same units as the map units (typically meter) for projected (planar) raster data. They should be in meter when the coordinate reference system is longitude/latitude.

For accuracy, always compute these values on the original data (do not first change the projection). Distances (needed for slope and aspect) for longitude/latitude data are computed on the WGS84 ellipsoid with Karney's algorithm.

Usage

```
## S4 method for signature 'SpatRaster'
terrain(x, v="slope", neighbors=8, unit="degrees", filename="", ...)
```

Arguments

<code>x</code>	SpatRaster, single layer with elevation values. Values should have the same unit as the map units, or in meters when the crs is longitude/latitude
<code>v</code>	character. One or more of these options: slope, aspect, TPI, TRI, TRIriley, TRIrmsd, roughness, flowdir (see Details)
<code>unit</code>	character. "degrees" or "radians" for the output of "slope" and "aspect"
<code>neighbors</code>	integer. Indicating how many neighboring cells to use to compute slope or aspect with. Either 8 (queen case) or 4 (rook case)
<code>filename</code>	character. Output filename
<code>...</code>	additional arguments for writing files as in writeRaster

Details

When `neighbors=4`, slope and aspect are computed according to Fleming and Hoffer (1979) and Ritter (1987). When `neighbors=8`, slope and aspect are computed according to Horn (1981). The Horn algorithm may be best for rough surfaces, and the Fleming and Hoffer algorithm may be better for smoother surfaces (Jones, 1997; Burrough and McDonnell, 1998).

If `slope = 0`, aspect is set to $0.5 \cdot \pi$ radians (or 90 degrees if `unit="degrees"`). When computing slope or aspect, the coordinate reference system of `x` must be known for the algorithm to differentiate between planar and longitude/latitude data.

`terrain` is not vectorized over "neighbors" or "unit" – only the first value is used.

`flowdir` returns the "flow direction" (of water), that is the direction of the greatest drop in elevation (or the smallest rise if all neighbors are higher). They are encoded as powers of 2 (0 to 7). The cell to the right of the focal cell is 1, the one below that is 2, and so on:

32	64	128
16	x	1
8	4	2

If two cells have the same drop in elevation, a random cell is picked. That is not ideal as it may prevent the creation of connected flow networks. ArcGIS implements the approach of Greenlee (1987) and I might adopt that in the future.

Most terrain indices are according to Wilson et al. (2007), as in [gdaldem](#). TRI (Terrain Ruggedness Index) is the mean of the absolute differences between the value of a cell and its 8 surrounding cells. TPI (Topographic Position Index) is the difference between the value of a cell and the mean value of its 8 surrounding cells. Roughness is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells.

TRIriley (TRI according to Riley et al. (2007)) returns the square root of summed squared differences between the value of a cell and its 8 surrounding cells. TRIrmsd computes the square root of the mean of the squared differences between these cells. The benefit of TRIrmsd

Such measures can also be computed with the [focal](#) function:

```
TRI <- focal(x, w=3, fun=\(x) sum(abs(x[-5]-x[5]))/8)
```

```
TPI <- focal(x, w=3, fun=\(x) x[5] - mean(x[-5]))
```

```
rough <- focal(x, w=3, fun=\(x) max(x) - min(x))
```

References

- Burrough, P., and R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.
- Fleming, M.D. and Hoffer, R.M., 1979. Machine processing of Landsat MSS data and DMA topographic data for forest cover type mapping. LARS Technical Report 062879. Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana.
- Horn, B.K.P., 1981. Hill shading and the reflectance map. Proceedings of the IEEE 69:14-47
- Jones, K.H., 1998. A comparison of algorithms used to compute hill slope as a property of the DEM. Computers & Geosciences 24: 315-323
- Karney, C.F.F., 2013. Algorithms for geodesics, J. Geodesy 87: 43-55. doi:10.1007/s00190-012-0578-z.
- Riley, S.J., De Gloria, S.D., Elliot, R. (1999): A Terrain Ruggedness that Quantifies Topographic Heterogeneity. Intermountain Journal of Science 5: 23-27.
- Ritter, P., 1987. A vector-based terrain and aspect generation algorithm. Photogrammetric Engineering and Remote Sensing 53: 1109-1111
- Wilson et al 2007, Multiscale Terrain Analysis of Multibeam Bathymetry Data for Habitat Mapping on the Continental Slope. Marine Geodesy 30:3-35

See Also

[viewshed](#)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
x <- terrain(r, "slope")
```

text

Add labels to a map

Description

Plots labels, that is a textual (rather than color) representation of values, on top an existing plot (map).

Usage

```
## S4 method for signature 'SpatRaster'
text(x, labels, digits=0, halo=FALSE, ...)

## S4 method for signature 'SpatVector'
text(x, labels, halo=FALSE, inside=FALSE, ...)
```

Arguments

x	SpatRaster or SpatVector
labels	character. Optional. Vector of labels with length(x) or a variable name from names(x)
digits	integer. How many digits should be used?
halo	logical. If TRUE a "halo" is printed around the text. If TRUE, additional arguments hc="white" and hw=0.1 can be modified to set the color and width of the halo
inside	logical. Should the text always be placed inside one the sub-geometries?
...	additional arguments to pass to graphics function text

See Also

[text](#), [plot](#), [halo](#)

Examples

```
r <- rast(nrows=4, ncols=4)
values(r) <- 1:ncell(r)
plot(r)
text(r)

plot(r)
text(r, halo=TRUE, hc="blue", col="white", hw=0.2)

plot(r, col=rainbow(16))
text(r, col=c("black", "white"), vfont=c("sans serif", "bold"), cex=2)
```

tighten	<i>tighten SpatRaster or SpatRasterDataset objects</i>
---------	--

Description

Combines data sources within a SpatRaster object (that are in memory, or from the same file) to allow for faster processing.

Or combine sub-datasets into a SpatRaster.

Usage

```
## S4 method for signature 'SpatRaster'
tighten(x)

## S4 method for signature 'SpatRasterDataset'
tighten(x)
```


Arguments

x SpatRaster or SpatRasterDataset

Value

SpatRaster

Examples

```
r <- rast(nrow=5, ncol=9, vals=1:45)
x <- c(r, r*2, r*3)
x
tighten(x)
```

time	<i>time of SpatRaster layers</i>
------	----------------------------------

Description

Get or set the time of the layers of a SpatRaster. Time can be stored as [POSIXlt](#) (date and time, with a resolution of seconds, and a time zone), [Date](#), "months", "years", or "yearmonths".

Usage

```
## S4 method for signature 'SpatRaster'
time(x, format="")

## S4 replacement method for signature 'SpatRaster'
time(x, tstep="")<-value

## S4 method for signature 'SpatRaster'
timeInfo(x)
```

Arguments

x SpatRaster

format One of "", "seconds" (POSIXlt), "days" (Date), "yearmonths" (decimal years), "years", "months". If "", the returned format is (based on) the format that was used to set the time

value Date, POSIXt, yearmon (defined in package zoo), or numeric

tstep One of "years", "months", "yearmonths". Used when value is numeric. Ignored when value is of type Date, POSIXt, or yearmon

Value

time: POSIXlt, Date, or numeric timeInfo: data.frame

See Also[depth](#)**Examples**

```
s <- rast(system.file("ex/logo.tif", package="terra"))

# Date"
d <- as.Date("2001-05-04") + 0:2
time(s) <- d
time(s)

# POSIX (date/time with a resolution of seconds)
time(s) <- as.POSIXlt(d)
time(s)

# with time zone
time(s) <- as.POSIXlt(Sys.time(), "America/New_York") + 0:2
time(s)
timeInfo(s)

# years
time(s, timestep="years") <- 2000 + 0:2
s

time(s, timestep="months") <- 1:3
s
```

tmpFiles

*Temporary files***Description**

List and optionally remove temporary files created by the terra package. These files are created when an output SpatRaster may be too large to store in memory (RAM). This can happen when no filename is provided to a function and when using functions where you cannot provide a filename.

Temporary files are automatically removed at the end of each R session that ends normally. You can use tmpFiles to see the files in the current sessions, including those that are orphaned (not connect to a SpatRaster object any more) and from other (perhaps old) sessions, and remove all the temporary files.

Usage

```
tmpFiles(current=TRUE, orphan=FALSE, old=FALSE, remove=FALSE)
```

Arguments

current	logical. If TRUE, temporary files from the current R session are included
orphan	logical. If TRUE, temporary files from the current R session that are no longer associated with a SpatRaster object (if current is TRUE these are also included)
old	logical. If TRUE, temporary files from other "R" sessions. Unless you are running multiple instances of R at the same time, these are from old (possibly crashed) R sessions and should be removed
remove	logical. If TRUE, temporary files are removed

Value

character

See Also

[terraOptions](#)

Examples

tmpFiles()

topology	<i>Vector topology methods</i>
----------	--------------------------------

Description

- makeNodes create nodes on lines
- mergeLines connect lines to form polygons
- removeDupNodes removes duplicate nodes in geometries and optionally rounds the coordinates
- emptyGeoms returns the indices of empty (null) geometries
- snap makes boundaries of geometries identical if they are very close to each other.

Usage

```
## S4 method for signature 'SpatVector'
mergeLines(x)
## S4 method for signature 'SpatVector'
snap(x, y=NULL, tolerance)
## S4 method for signature 'SpatVector'
removeDupNodes(x, digits = -1)
## S4 method for signature 'SpatVector'
makeNodes(x)
```

Arguments

x	SpatVector of lines or polygons
y	SpatVector of lines or polygons to snap to. If NULL snapping is to the other geometries in x
tolerance	numeric. Snapping tolerance (distance between geometries)
digits	numeric. Number of digits used in rounding. Ignored if < 0

Value

SpatVector

See Also

[sharedPaths](#), [gaps](#), [simplifyGeom](#)

Examples

```
p1 <- as.polygons(ext(0,1,0,1))
p2 <- as.polygons(ext(1.1,2,0,1))

p <- rbind(p1, p2)

y <- snap(p, tol=.15)
plot(p, lwd=3, col="light gray")
lines(y, col="red", lwd=2)
```

transpose	<i>Transpose</i>
-----------	------------------

Description

Transpose a SpatRaster

Usage

```
## S4 method for signature 'SpatRaster'
t(x)

## S4 method for signature 'SpatVector'
t(x)

## S4 method for signature 'SpatRaster'
trans(x, filename="", ...)
```

Arguments

x	SpatRaster
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

See Also

[flip](#), [rotate](#)

Examples

```
r <- rast(nrows=18, ncols=36)
values(r) <- 1:ncell(r)
tr1 <- t(r)
tr2 <- trans(r)
ttr <- trans(tr2)
```

trim

Trim a SpatRaster

Description

Trim (shrink) a SpatRaster by removing outer rows and columns that are NA or another value.

Usage

```
## S4 method for signature 'SpatRaster'
trim(x, padding=0, value=NA, filename="", ...)
```

Arguments

x	SpatRaster
padding	integer. Number of outer rows/columns to keep
value	numeric. The value of outer rows or columns that are to be removed
filename	character. Output filename
...	additional arguments for writing files as in writeRaster

Value

SpatRaster

Examples

```

r <- rast(ncols=10, nrows=10, xmin=0,xmax=10,ymin=0,ymax=10)
v <- rep(NA, ncell(r))
v[c(12,34,69)] <- 1:3
values(r) <- v
s <- trim(r)

```

union

*Union SpatVector or SpatExtent objects***Description**

Overlapping polygons (between, not within, objects) are intersected. Union for lines and points simply combines the two data sets; without any geometric intersections. This is equivalent to [c](#). Attributes are joined. See [c](#) if you want to combine polygons without intersection.

If *x* and *y* have a different geometry type, a *SpatVectorCollection* is returned.

If a single *SpatVector* is supplied, overlapping polygons are intersected. Original attributes are lost. New attributes allow for determining how many, and which, polygons overlapped.

SpatExtent: Objects are combined into their union; this is equivalent to `+`.

Usage

```

## S4 method for signature 'SpatVector,SpatVector'
union(x, y)

## S4 method for signature 'SpatVector,missing'
union(x, y)

## S4 method for signature 'SpatExtent,SpatExtent'
union(x, y)

```

Arguments

<i>x</i>	<i>SpatVector</i> or <i>SpatExtent</i>
<i>y</i>	Same as <i>x</i> or missing

Value

SpatVector or *SpatExtent*

See Also

[intersect](#)

[merge](#) and [mosaic](#) to union *SpatRaster* objects.

[crop](#) and [extend](#) for the union of *SpatRaster* and *SpatExtent*.

[merge](#) for merging a *data.frame* with attributes of a *SpatVector*.

[aggregate](#) to dissolve *SpatVector* objects.

Examples

```
e1 <- ext(-10, 10, -20, 20)
e2 <- ext(0, 20, -40, 5)
union(e1, e2)

#SpatVector
v <- vect(system.file("ex/lux.shp", package="terra"))
v <- v[,3:4]
p <- vect(c("POLYGON ((5.8 49.8, 6 49.9, 6.15 49.8, 6 49.65, 5.8 49.8))",
"POLYGON ((6.3 49.9, 6.2 49.7, 6.3 49.6, 6.5 49.8, 6.3 49.9))"), crs=crs(v))
values(p) <- data.frame(pid=1:2, value=expanses(p))
u <- union(v, p)
plot(u, "pid")

b <- buffer(v, 1000)

u <- union(b)
u$sum <- rowSums(as.data.frame(u))
plot(u, "sum")
```

unique	<i>Unique values</i>
--------	----------------------

Description

This method returns the unique values in a SpatRaster, or removes duplicates records (geometry and attributes) in a SpatVector.

Usage

```
## S4 method for signature 'SpatRaster'
unique(x, incomparables=FALSE, digits=NA, na.rm=TRUE, as.raster=FALSE)

## S4 method for signature 'SpatVector'
unique(x, incomparables=FALSE, ...)
```

Arguments

x	SpatRaster or SpatVector
incomparables	logical. If FALSE and x is a SpatRaster: the unique values are determined for all layers together, and the result is a matrix. If TRUE, each layer is evaluated separately, and a list is returned. If x is a SpatVector this argument is as for a data.frame
digits	integer. The number of digits for rounding the values before finding the unique values. Use NA means to not do any rounding
na.rm	logical. If TRUE, NaN is included if there are any missing values
as.raster	logical. If TRUE, a single-layer categorical SpatRaster with the unique values is returned
...	additional arguments passed on to unique

Value

If x is a `SpatRaster`: `data.frame` or `list` (if `incomparables=FALSE`)

If x is a `SpatVector`: `SpatVector`

Examples

```
r <- rast(ncols=5, nrows=5)
values(r) <- rep(1:5, each=5)
unique(r)
s <- c(r, round(r/3))
unique(s)
unique(s,TRUE)

unique(s, as.raster=TRUE)

v <- vect(cbind(x=c(1:5,1:5), y=c(5:1,5:1)),
crs="+proj=utm +zone=1 +datum=WGS84")
nrow(v)
u <- unique(v)
nrow(u)

values(v) <- c(1:5, 1:3, 5:4)
unique(v)
```

units

units of `SpatRaster` or `SpatRasterDataSet`

Description

Get or set the units of the layers of a `SpatRaster` or the datasets in a `SpatRasterDataSet`.

Usage

```
## S4 method for signature 'SpatRaster'
units(x)

## S4 replacement method for signature 'SpatRaster'
units(x)<-value

## S4 method for signature 'SpatRasterDataset'
units(x)

## S4 replacement method for signature 'SpatRasterDataset'
units(x)<-value
```


Arguments

x	SpatRaster
value	character

Value

character

See Also

[time](#), [names](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))

units(s) <- c("m/s", "kg", "ha")
units(s)
s

units(s) <- "kg"
units(s)
```

update	<i>Change values in a file</i>
--------	--------------------------------

Description

Change the contents of a file that is the data source of a SpatRaster. BE CAREFUL as you are overwriting values in an existing file.

Usage

```
## S4 method for signature 'SpatRaster'
update(object, names=FALSE, crs=FALSE, extent=FALSE)
```

Arguments

object	SpatRaster
names	logical. Should the names be updated?
crs	logical. Should the coordinate reference system be updated?
extent	logical. Should the extent be updated?

Value

SpatRaster (invisibly)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
fname <- paste0(tempfile(), ".tif")
x <- writeRaster(s, fname)
names(x) <- c("A", "B", "C")
ext(x) <- ext(x) + 1
crs(x) <- "+proj=utm +zone=1"

update(x, names=TRUE, crs=TRUE, extent=TRUE)

rast(fname)
```

valid	<i>Check or fix polygon or extent validity</i>
-------	--

Description

Check the validity of polygons or attempt to fix it. Or check the validity of a SpatExtent.

Usage

```
## S4 method for signature 'SpatVector'
is.valid(x, messages=FALSE, as.points=FALSE)

## S4 method for signature 'SpatVector'
makeValid(x)

## S4 method for signature 'SpatExtent'
is.valid(x)
```

Arguments

x	SpatVector
messages	logical. If TRUE the error messages are returned
as.points	logical. If TRUE, it is attempted to return locations where polygons are invalid as a SpatVector or points

Value

logical

Examples

```
w <- vect("POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")
is.valid(w)

w <- vect("POLYGON ((0 -5, 10 0, 10 -10, 4 -2, 0 -5))")
is.valid(w)
```

```
is.valid(w, TRUE)

plot(w)
points(cbind(4.54, -2.72), cex=2, col="red")
```

values

*Cell values and geometry attributes***Description**

Get the cell values of a `SpatRaster` or the attributes of a `SpatVector`.

By default all values returned are numeric. This is because a vector or matrix can only store one data type, and a `SpatRaster` may consist of multiple data types. However, if all layers have integer or logical values, the returned values also have that datatype.

Note that with `values(x, dataframe=TRUE)` and `as.data.frame(x)` the values returned match the type of each layer, and can be a mix of numeric, logical, integer, and factor.

Usage

```
## S4 method for signature 'SpatRaster'
values(x, mat=TRUE, dataframe=FALSE, row=1,
       nrow=nrow(x), col=1, ncol=ncol(x), na.rm=FALSE, ...)

## S4 method for signature 'SpatVector'
values(x, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>mat</code>	logical. If <code>TRUE</code> , values are returned as a matrix instead of as a vector, except when <code>dataframe</code> is <code>TRUE</code>
<code>dataframe</code>	logical. If <code>TRUE</code> , values are returned as a <code>data.frame</code> instead of as a vector (also if matrix is <code>TRUE</code>)
<code>row</code>	positive integer. Row number to start from, should be between 1 and <code>nrow(x)</code>
<code>nrow</code>	positive integer. How many rows?
<code>col</code>	positive integer. Column number to start from, should be between 1 and <code>ncol(x)</code>
<code>ncol</code>	positive integer. How many columns? Default is the number of columns left after the start column
<code>na.rm</code>	logical. Remove NAs?
<code>...</code>	additional arguments passed to <code>data.frame</code>

Details

If `x` is a `SpatRaster`, and `mat=FALSE`, the values are returned as a vector. In cell-order by layer. If `mat=TRUE`, a matrix is returned in which the values of each layer are represented by a column (with `ncell(x)` rows). The values per layer are in cell-order, that is, from top-left, to top-right and then down by row. Use `as.matrix(x, wide=TRUE)` for an alternative matrix representation where the number of rows and columns matches that of `x`.

Value

matrix or data.frame

See Also

`values<-`, `focalValues`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
r
x <- values(r)
x[3650:3655, ]
r[3650:3655]
```

```
ff <- system.file("ex/lux.shp", package="terra")
v <- vect(ff)
y <- values(v)
head(y)
```

vect

Create SpatVector objects

Description

Methods to create a `SpatVector` from a filename or other R object.

A filename can be for a shapefile or any spatial file format.

You can use a data.frame to make a `SpatVector` of points; or a "geom" matrix to make a `SpatVector` of any supported geometry (see examples and [geom](#)).

You can supply a list of `SpatVectors` to append them into a single `SpatVector`.

`SpatVectors` can also be created from "Well Known Text", and from spatial vector data objects defined in the `sf` or `sp` packages.

Usage

```
## S4 method for signature 'character'
vect(x, layer="", query="", extent=NULL, filter=NULL,
     crs="", proxy=FALSE, what="")

## S4 method for signature 'matrix'
vect(x, type="points", atts=NULL, crs="")

## S4 method for signature 'data.frame'
vect(x, geom=c("lon", "lat"), crs="", keepgeom=FALSE)

## S4 method for signature 'list'
vect(x, type="points", crs="")

## S4 method for signature 'SpatExtent'
vect(x, crs="")

## S4 method for signature 'SpatVectorCollection'
vect(x)

## S4 method for signature 'sf'
vect(x)
```

Arguments

x	character. A filename; or a "Well Known Text" string; SpatExtent, data.frame (to make a SpatVector of points); a "geom" matrix to make a SpatVector of any supported geometry (see examples and geom); a spatial vector data object defined in the sf or sp packages; or a list with matrices with coordinates
layer	character. layer name to select a layer from a file (database) with multiple layers
query	character. An query to subset the dataset in the OGR-SQL dialect
extent	Spat* object. The extent of the object is used as a spatial filter to select the geometries to read. Ignored if filter is not NULL
filter	SpatVector. Used as a spatial filter to select geometries to read (the convex hull is used for lines or points). It is guaranteed that all features that overlap with the extent of filter will be returned. It can happen that additional geometries are returned
type	character. Geometry type. Must be "points", "lines", or "polygons"
atts	data.frame with the attributes. The number of rows must match the number of geometrical elements
crs	character. The coordinate reference system in one of the following formats: WKT/WKT2, <authority>:<code>, or PROJ-string notation (see crs). See crs
proxy	logical. If TRUE a SpatVectorProxy is returned
what	character indicating what to read. Either "" for geometries and attributes, geoms to only read the geometries, attributes to only read the attributes (that are returned as a data.frame)

geom	character. The field name(s) with the geometry data. Either two names for x and y coordinates of points, or a single name for a single column with WKT geometries)
keepgeom	logical. If TRUE the geom variable(s) is (are) also included in the attributes

Value

SpatVector

See Also[geom](#)**Examples**

```
### SpatVector from file
f <- system.file("ex/lux.shp", package="terra")
f
v <- vect(f)
v

## subsetting (large) files
## with attribute query
v <- vect(f, query="SELECT NAME_1, NAME_2, ID_2 FROM lux WHERE ID_2 < 4")
v

## with an extent
e <- ext(5.9, 6.3, 49.9, 50)
v <- vect(f, extent=e)

## with polygons
p <- as.polygons(e)
v <- vect(f, filter=p)

### SpatVector from a geom matrix
x1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x3 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
hole <- rbind(c(80,0), c(105,13), c(120,2), c(105,-13))
z <- rbind(cbind(object=1, part=1, x1, hole=0), cbind(object=2, part=1, x3, hole=0),
cbind(object=3, part=1, x2, hole=0), cbind(object=3, part=1, hole, hole=1))
colnames(z)[3:4] <- c('x', 'y')

p <- vect(z, "polygons")
p

z[z[, "hole"]==1, "object"] <- 4
lns <- vect(z[,1:4], "lines")
plot(p)
lines(lns, col="red", lwd=2)
```

```

### from wkt
v <- vect("POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")

wkt <- c("MULTIPOLYGON ( ((40 40, 20 45, 45 30, 40 40)),
((20 35, 10 30, 10 10, 30 5, 45 20, 20 35),(30 20, 20 15, 20 25, 30 20)))",
"POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")
w <- vect(wkt)

# combine two SpatVectors
vw <- rbind(w, v)

# add a data.frame
d <- data.frame(id=1:2, name=c("a", "b"))
values(w) <- d

# add data.frame on creation, here from a geom matrix
g <- geom(w)
d <- data.frame(id=1:2, name=c("a", "b"))
m <- vect(g, "polygons", atts=d, crs="+proj=longlat +datum=WGS84")

### SpatVector from a data.frame
d$wkt <- wkt
x <- vect(d, geom="wkt")

d$wkt <- NULL
d$lon <- c(0,10)
d$lat <- c(0,10)
x <- vect(d, geom=c("lon", "lat"))

# SpatVector to sf
#sf::st_as_sf(x)

```

vector_layers

List or remove layers from a vector file

Description

List or remove layers from a vector file that supports layers such as GPKG

Usage

```
vector_layers(filename, delete="", return_error=FALSE)
```

Arguments

filename	character. filename
delete	character. layers to be deleted (ignored if the value is "")
return_error	logical. If TRUE, an error occurs if some layers cannot be deleted. Otherwise a warning is given

viewshed

*Compute a viewshed***Description**

Use elevation data to compute the locations that can be seen, or how much higher they would have to be to be seen, from a certain position. The raster data coordinate reference system must planar (not lon/lat), with the elevation values in the same unit as the distance unit of the coordinate reference system.

Usage

```
## S4 method for signature 'SpatRaster'
viewshed(x, loc, observer=1.80, target=0, curvcoef=6/7, output="yes/no", filename="", ...)
```

Arguments

x	SpatRaster, single layer with elevation values. Values should have the same unit as the map units
loc	location (x and y coordinates) or a cell number
observer	numeric. The height above the elevation data of the observer
target	numeric. The height above the elevation data of the targets
curvcoef	numeric. Coefficient to consider the effect of the curvature of the earth and refraction of the atmosphere. The elevation values are corrected with: $\text{elevation} = \text{elevation} - \text{curvcoef} * (\text{distance})^2 / (\text{earth_diameter})$. This means that with the default value of 0.85714, you lose sight of about 1 meter of elevation for each 385 m of planar distance
output	character. Can be "yes/no" to get a binary (logical) output showing what areas are visible; "land" to get the height above the current elevation that would be visible; or "sea" the elevation above sea level that would be visible
filename	character. Output filename
...	Options for writing files as in writeRaster

References

The algorithm used is by Wang et al.: https://www.asprs.org/wp-content/uploads/pers/2000journal/january/2000_jan_87-90.pdf.

See Also

[terrain](#)

Examples

```
if (gdal() >= 3.1) {
  f <- system.file("ex/elev.tif", package="terra")
  r <- rast(f)
  x <- project(r, "EPSG:2169")
  p <- cbind(70300, 96982)
  v <- viewshed(x, p, 0, 0, 0.85714)
}
```

voronoi

Voronoi diagram and Delaunay triangles

Description

Get a Voronoi diagram or Delaunay triangles for points, or the nodes of lines or polygons

Usage

```
## S4 method for signature 'SpatVector'
voronoi(x, bnd=NULL, tolerance=0, as.lines=FALSE, deldir=FALSE)

## S4 method for signature 'SpatVector'
delaunay(x, tolerance=0, as.lines=FALSE)
```

Arguments

x	SpatVector
bnd	SpatVector to set the outer boundary of the voronoi diagram
tolerance	numeric >= 0, snapping tolerance (0 is no snapping)
as.lines	logical. If TRUE, lines are returned without the outer boundary
deldir	logical. If TRUE, the deldir is used instead of the GEOS C++ library method. It has been reported that deldir does not choke on very large data sets

Value

SpatVector

Examples

```
wkt <- c("MULTIPOLYGON ( ((40 40, 20 45, 45 30, 40 40)),
  ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35),(30 20, 20 15, 20 25, 30 20)))",
  "POLYGON ((0 -5, 10 0, 10 -10, 0 -5))")
x <- vect(wkt)
v <- voronoi(x)
v
```

```
d <- delaunay(x)
d

plot(v, lwd=2, col=rainbow(15))
lines(x, col="gray", lwd=2)
points(x)
```

vrt

Virtual Raster Dataset

Description

Create a Virtual Raster Dataset (VRT) from a collection of file-based raster datasets (tiles).

Usage

```
## S4 method for signature 'character'
vrt(x, filename="", options=NULL, overwrite=FALSE)
```

Arguments

x	character. Filenames of raster "tiles". See tiles
filename	character. Output VRT filename
options	character. All arguments as separate vector elements. Options as for gdalbuildvrt
overwrite	logical. Should filename be overwritten if it exists?

Value

SpatRaster

Note

A VRT can reference very many datasets. These are not all opened at the same time. The default is to open not more than 100 files. To increase performance, this maximum limit can be increased by setting the GDAL_MAX_DATASET_POOL_SIZE configuration option to a bigger value with [setGDALconfig](#). Note that a typical user process on Linux is limited to 1024 simultaneously opened files.

See Also

[makeTiles](#) to create tiles; [makeVRT](#) to create a .vrt file for a binary raster file that does not have a header file.

Examples

```
r <- rast(ncols=100, nrows=100)
values(r) <- 1:ncell(r)
x <- rast(ncols=2, nrows=2)
filename <- paste0(tempfile(), "_tif")
ff <- makeTiles(r, x, filename)
ff

#vrtfile <- paste0(tempfile(), ".vrt")
#v <- vrt(ff, vrtfile)

## output in lower resolution
#vrtfile <- paste0(tempfile(), ".vrt")
#v <- vrt(ff, vrtfile, options = c("-tr", 5, 5))
#head(readLines(vrtfile))
#v
```

weighted.mean	<i>Weighted mean of layers</i>
---------------	--------------------------------

Description

Compute the weighted mean for each cell of the layers of a `SpatRaster`. The weights can be spatially variable or not.

Usage

```
## S4 method for signature 'SpatRaster,numeric'
weighted.mean(x, w, na.rm=FALSE, filename="", ...)

## S4 method for signature 'SpatRaster,SpatRaster'
weighted.mean(x, w, na.rm=FALSE, filename="", ...)
```

Arguments

x	<code>SpatRaster</code>
w	A vector of weights (one number for each layer), or for spatially variable weights, a <code>SpatRaster</code> with weights (should have the same extent, resolution and number of layers as x)
na.rm	Logical. Should missing values be removed?
filename	character. Output filename
...	options for writing files as in writeRaster

Value

`SpatRaster`

See Also

[Summary-methods](#), [weighted.mean](#)

Examples

```
b <- rast(system.file("ex/logo.tif", package="terra"))

# give least weight to first layer, most to last layer
wm1 <- weighted.mean(b, w=1:3)

# spatially varying weights
# weigh by column number
w1 <- init(b, "col")

# weigh by row number
w2 <- init(b, "row")
w <- c(w1, w2, w2)

wm2 <- weighted.mean(b, w=w)
```

where

Where are the cells with the min or max values?

Description

This method returns the cell numbers for the cells with the min or max values of each layer in a `SpatRaster`.

Usage

```
## S4 method for signature 'SpatRaster'
where.min(x, values=TRUE, list=FALSE)

## S4 method for signature 'SpatRaster'
where.max(x, values=TRUE, list=FALSE)
```

Arguments

<code>x</code>	<code>SpatRaster</code>
<code>values</code>	logical. If TRUE the min or max values are also returned
<code>list</code>	logical. If TRUE a list is returned instead of a matrix

Value

matrix or list

See Also

[which](#) and [Summary-methods](#) for `which.min` and `which.max`

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
where.min(r)
```

which.lyr	<i>Which cells are TRUE?</i>
-----------	------------------------------

Description

This method returns a single layer SpatRaster with cell values indicating the first layer in the input that is TRUE. All numbers that are not zero (or FALSE), are considered to be TRUE.

Usage

```
## S4 method for signature 'SpatRaster'
which.lyr(x)
```

Arguments

x SpatRaster

Value

SpatRaster

See Also

[isTRUE](#), [which](#), See [Summary-methods](#) for [which.min](#) and [which.max](#)

Examples

```
s <- rast(system.file("ex/logo.tif", package="terra"))
x <- which.lyr(s > 100)
```

width

SpatVector geometric properties**Description**

`width` returns the minimum diameter of the geometry, defined as the smallest band that contains the geometry, where a band is a strip of the plane defined by two parallel lines. This can be thought of as the smallest hole that the geometry can be moved through, with a single rotation.

`clearance` returns the minimum clearance of a geometry. The minimum clearance is the smallest amount by which a vertex could be moved to produce an invalid polygon, a non-simple linestring, or a multipoint with repeated points. If a geometry has a minimum clearance of 'mc', it can be said that:

No two distinct vertices in the geometry are separated by less than 'mc'. No vertex is closer than 'mc' to a line segment of which it is not an endpoint. If the minimum clearance cannot be defined for a geometry (such as with a single point, or a multipoint whose points are identical, NA is returned.

Usage

```
## S4 method for signature 'SpatVector'
width(x, as.lines=FALSE)
## S4 method for signature 'SpatVector'
clearance(x, as.lines=FALSE)
```

Arguments

<code>x</code>	<code>SpatVector</code> of lines or polygons
<code>as.lines</code>	logical. If TRUE lines are returned that define the width or clearance

Value

numeric or `SpatVector`

See Also

[minRect](#)

Examples

```
f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
width(v)
clearance(v)
```

window*Set a window*

Description

Assign a window (area of interest) to a `SpatRaster` with a `SpatExtent`, or set it to `NULL` to remove the window. This is similar to [crop](#) without actually creating a new dataset.

The window is intersect with the extent of the `SpatRaster`. It is envisioned that in a future version, the window may also go outside these boundaries.

Usage

```
## S4 replacement method for signature 'SpatRaster'
window(x)<-value

## S4 method for signature 'SpatRaster'
window(x)
```

Arguments

x	<code>SpatRaster</code>
value	<code>SpatExtent</code>

Value

none for `window<-` and logical for `window`

See Also

[crop](#), [extend](#)

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
global(r, "mean", na.rm=TRUE)
e <- ext(c(5.9, 6, 49.95, 50))

window(r) <- e
global(r, "mean", na.rm=TRUE)
r

x <- rast(f)
xe <- crop(x, e)
global(xe, "mean", na.rm=TRUE)

b <- c(xe, r)
```

```

window(b)
b

window(r) <- NULL
r

```

wrap

wrap and unwrap

Description

Use `wrap` to pack a `SpatRaster` or `SpatVector` object to create a `Packed*` object. Packed objects can be passed over a connection that serializes (e.g. to nodes on a computer cluster). At the receiving end they need to be unpacked with `unwrap`.

Usage

```

## S4 method for signature 'SpatRaster'
wrap(x, proxy=FALSE)

## S4 method for signature 'SpatVector'
wrap(x)

## S4 method for signature 'ANY'
unwrap(x)

```

Arguments

<code>x</code>	<code>SpatVector</code> or <code>SpatRaster</code>
<code>proxy</code>	logical. If <code>FALSE</code> raster cell values are forced to memory if possible. If <code>TRUE</code> , a reference to source filenames is stored for data sources that are not in memory

Value

`wrap`: `Packed*` object

`unwrap`: `SpatRaster` or `SpatVector`

Examples

```

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)
p <- wrap(v)
p
vv <- vect(p)
vv

```


writeCDF

*Write raster data to a NetCDF file***Description**

Write a `SpatRaster` or `SpatRasterDataset` to a NetCDF file.

When using a `SpatRasterDataset`, the `varname`, `longname`, and `unit` should be set in the object (see examples).

Always use the ".nc" or ".cdf" file extension to assure that the file can be properly read again by GDAL

Usage

```
## S4 method for signature 'SpatRaster'
writeCDF(x, filename, varname, longname="", unit="", split=FALSE, ...)

## S4 method for signature 'SpatRasterDataset'
writeCDF(x, filename, overwrite=FALSE, zname="time", atts="",
        gridmap="", prec="float", compression=NA, missval, ...)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatRasterDataset</code>
<code>filename</code>	character. Output filename
<code>varname</code>	character. Name of the dataset
<code>longname</code>	character. Long name of the dataset
<code>unit</code>	character. Unit of the data
<code>split</code>	logical. If TRUE each layer of <code>x</code> is treated as a sub-dataset
<code>atts</code>	character. A vector of additional global attributes to write. The must be formatted like <code>c("x=a value", "y=abc")</code>
<code>gridmap</code>	character. The crs is always writing to the file in standard formats. With this argument you can also write the format commonly used in netcdf files. Something like <code>c("grid_mapping_name=lambert_azimuthal_equal_area", "longitude_of_projection_origin=10", "latitude_of_projection_origin=52", "false_easting=4321000", "false_northing=3210000")</code>
<code>overwrite</code>	logical. If TRUE, <code>filename</code> is overwritten
<code>zname</code>	character. The name of the "time" dimension
<code>prec</code>	character. One of "double", "float", "integer", "short", "byte" or "char"
<code>compression</code>	Can be set to an integer between 1 (least compression) and 9 (most compression)
<code>missval</code>	numeric, the number used to indicate missing values
<code>...</code>	additional arguments passed on to the <code>SpatRasterDataset</code> method, and from there possibly to <code>ncvar_def</code>

Value

SpatRaster or SpatDataSet

See Also

see [writeRaster](#) for writing other file formats

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
fname <- paste0(tempfile(), ".nc")
#rr <- writeCDF(r, fname, overwrite=TRUE, varname="alt",
#             longname="elevation in m above sea level", unit="m")

a <- rast(ncols=5, nrows=5, nl=50)
values(a) <- 1:prod(dim(a))
time(a) <- as.Date("2020-12-31") + 1:nlyr(a)
#aa <- writeCDF(a, fname, overwrite=TRUE, varname="power",
#             longname="my nice data", unit="U/Pa")

b <- sqrt(a)
s <- sds(a, b)
names(s) <- c("temp", "prec")
longnames(s) <- c("temperature (C)", "precipitation (mm)")
units(s) <- c("C", "mm")
#ss <- writeCDF(s, fname, overwrite=TRUE)

# for CRAN
#file.remove(fname)
```

writeRaster	<i>Write raster data to a file</i>
-------------	------------------------------------

Description

Write a SpatRaster object to a file.

Usage

```
## S4 method for signature 'SpatRaster,character'
writeRaster(x, filename, overwrite=FALSE, ...)
```

Arguments

x	SpatRaster
filename	character. Output filename. Can be a single filename, or as many filenames as nlyr(x) to write a file for each layer
overwrite	logical. If TRUE, filename is overwritten
...	additional arguments for for writing files. See Details

Details

In writeRaster, and in other methods that generate SpatRaster objects, options for writing raster files to disk can be provided as additional arguments or, in a few cases, as the wopt argument (a named list) if the additional arguments are already used for a different purpose. See [terraOptions](#) to get or set default values. The following options are available:

name	description
datatype	values accepted are "INT1U", "INT2U", "INT2S", "INT4U", "INT4S", "FLT4S", "FLT8S". With GDAL >= 3.5 y
filetype	file format expresses as GDAL driver names . If this argument is not supplied, the driver is derived from the filename
gdal	GDAL driver specific datasource creation options. See the GDAL documentation. For example, with the GeoTiff
tempdir	the path where temporary files are to be written to.
progress	positive integer. If the number of chunks is larger, a progress bar is shown.
memfrac	numeric between 0 and 0.9 (higher values give a warning). The fraction of available RAM that terra is allowed to
memmax	memmax - the maximum amount of RAM (in GB) that terra can use when processing a raster dataset. Should be
names	output layer names.
NAflag	numeric. value to represent missing (NA or NaN) values. See note
scale	numeric. Cell values written to disk are divided by this value (default is 1). See scoff
offset	numeric. Value that is subtracted from the cell values written to disk (default is 0). See scoff
verbose	logical. If TRUE debugging information is printed
steps	positive integers. In how many steps (chunks) do you want to process the data (for debugging)
todisk	logical. If TRUE processing operates as if the dataset is very large and needs to be written to a temporary file (for c

Value

SpatRaster. This function is used for the side-effect of writing values to a file.

Note

GeoTiff files are, by default, written with LZW compression. If you do not want compression, use `gdal="COMPRESS=NONE"`.

When writing integer values the lowest available value (given the datatype) is used to represent NA for signed types, and the highest value is used for unsigned values. This can be a problem with byte data (between 0 and 255) as the value 255 is reserved for NA. To keep the value 255, you need to set another value as NAflag, or do not set a NAflag (with NAflag=NA)

See Also

see [writeCDF](#) for writing NetCDF files.

Examples

```
r <- rast(nrows=5, ncols=5, vals=1:25)

# create a temporary filename for the example
f <- file.path(tempdir(), "test.tif")

writeRaster(r, f, overwrite=TRUE)
```

```

writeRaster(r, f, overwrite=TRUE, gdal=c("COMPRESS=NONE", "TFW=YES"), datatype='INT1U')

## Or with a wopt argument:

writeRaster(r, f, overwrite=TRUE, wopt= list(gdal=c("COMPRESS=NONE"), datatype='INT1U'))

## remove the file
unlink(f)

```

writeVector

Write SpatVector data to a file

Description

Write a SpatVector to a file. You can choose one of many file formats.

Usage

```

## S4 method for signature 'SpatVector,character'
writeVector(x, filename, filetype=NULL, layer=NULL, insert=FALSE,
            overwrite=FALSE, options="ENCODING=UTF-8")

```

Arguments

x	SpatVector
filename	character. Output filename
filetype	character. A file format associated with a GDAL "driver" such as "ESRI Shapefile". See <code>gdal(drivers=TRUE)</code> or the GDAL docs . If NULL it is attempted to guess the filetype from the filename extension
layer	character. Output layer name. If NULL the filename is used
insert	logical. If TRUE, a new layer is inserted into the file, if the format allows it (e.g. GPKG allows that). See vector_layers to remove a layer
overwrite	logical. If TRUE, filename is overwritten
options	character. Format specific GDAL options such as "ENCODING=UTF-8". Use NULL or "" to not use any options

Examples

```

v <- vect(cbind(1:5,1:5))
crs(v) <- "+proj=longlat +datum=WGS84"
v$id <- 1:length(v)
v$name <- letters[1:length(v)]
tmpf1 <- paste0(tempfile(), ".gpkg")
writeVector(v, tmpf1)
x <- vect(tmpf1)

f <- system.file("ex/lux.shp", package="terra")

```

```
v <- vect(f)
tmpf2 <- paste0(tempfile(), ".gpkg")
writeVector(v, tmpf2)
y <- vect(tmpf2)
```

xmin

Get or set single values of an extent

Description

Get or set single values of an extent. Values can be set for a `SpatExtent` or `SpatRaster`, but not for a `SpatVector`

Usage

```
## S4 method for signature 'SpatExtent'
xmin(x)
```

```
## S4 method for signature 'SpatExtent'
xmax(x)
```

```
## S4 method for signature 'SpatExtent'
ymin(x)
```

```
## S4 method for signature 'SpatExtent'
ymax(x)
```

```
## S4 method for signature 'SpatRaster'
xmin(x)
```

```
## S4 method for signature 'SpatRaster'
xmax(x)
```

```
## S4 method for signature 'SpatRaster'
ymin(x)
```

```
## S4 method for signature 'SpatRaster'
ymax(x)
```

```
## S4 method for signature 'SpatVector'
xmin(x)
```

```
## S4 method for signature 'SpatVector'
xmax(x)
```

```
## S4 method for signature 'SpatVector'
ymin(x)
```

```
## S4 method for signature 'SpatVector'
ymax(x)

## S4 replacement method for signature 'SpatRaster,numeric'
xmin(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
xmax(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
ymin(x)<-value

## S4 replacement method for signature 'SpatRaster,numeric'
ymax(x)<-value
```

Arguments

x	SpatRaster, SpatExtent, or SpatVector
value	numeric

Value

SpatExtent or numeric coordinate

Examples

```
r <- rast()
ext(r)
ext(c(0, 20, 0, 20))

xmin(r)
xmin(r) <- 0
xmin(r)
```

xyRowColCell

Coordinates from a row, column or cell number and vice versa

Description

Get coordinates of the center of raster cells for a row, column, or cell number of a SpatRaster object. Or get row, column, or cell numbers from coordinates or from each other.

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the SpatRaster object. row numbers start at 1 at the top, column numbers start at 1 at the left.

Usage

```
## S4 method for signature 'SpatRaster,numeric'
xFromCol(object, col)

## S4 method for signature 'SpatRaster,numeric'
yFromRow(object, row)

## S4 method for signature 'SpatRaster,numeric'
xyFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
xFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
yFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
colFromX(object, x)

## S4 method for signature 'SpatRaster,numeric'
rowFromY(object, y)

## S4 method for signature 'SpatRaster,numeric,numeric'
cellFromRowCol(object, row, col)

## S4 method for signature 'SpatRaster,numeric,numeric'
cellFromRowColCombine(object, row, col)

## S4 method for signature 'SpatRaster,numeric,numeric'
rowColCombine(object, row, col)

## S4 method for signature 'SpatRaster,numeric'
rowFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
colFromCell(object, cell)

## S4 method for signature 'SpatRaster,numeric'
rowColFromCell(object, cell)

## S4 method for signature 'SpatRaster,matrix'
cellFromXY(object, xy)
```

Arguments

object	SpatRaster
cell	integer. cell number(s)
col	integer. column number(s) or missing (equivalent to all columns)

row	integer. row number(s) or missing (equivalent to all rows)
x	x coordinate(s)
y	y coordinate(s)
xy	matrix of x and y coordinates

Details

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the SpatRaster (see [ncell](#)).

Value

xFromCol, yFromCol, xFromCell, yFromCell: vector of x or y coordinates

xyFromCell: matrix(x,y) with coordinate pairs

colFromX, rowFromY, cellFromXY, cellFromRowCol, rowFromCell, colFromCell: vector of row, column, or cell numbers

rowColFromCell, rowColCombine: matrix of row and column numbers

See Also

[crds](#)

Examples

```
r <- rast()

xFromCol(r, c(1, 120, 180))
yFromRow(r, 90)
xyFromCell(r, 10000)
xyFromCell(r, c(0, 1, 32581, ncell(r), ncell(r)+1))

cellFromRowCol(r, 5, 5)
cellFromRowCol(r, 1:2, 1:2)
cellFromRowCol(r, 1, 1:3)

# all combinations
cellFromRowColCombine(r, 1:2, 1:2)

colFromX(r, 10)
rowFromY(r, 10)
xy <- cbind(lon=c(10,5), lat=c(15, 88))
cellFromXY(r, xy)

# if no row/col specified all are returned
range(xFromCol(r))
length(yFromRow(r))
```

zonal	<i>Zonal statistics</i>
-------	-------------------------

Description

Compute zonal statistics, that is summarize values of a `SpatRaster` for each "zone" defined by another `SpatRaster`, or by a `SpatVector` with polygon geometry.

If `fun` is a true R function, the `<SpatRaster,SpatRaster>` method may fail when using very large `SpatRasters`, except for the functions ("mean", "min", "max", "sum", "isNA", and "notNA").

You can also summarize values of a `SpatVector` for each polygon (zone) defined by another `SpatVector`.

Usage

```
## S4 method for signature 'SpatRaster,SpatRaster'
zonal(x, z, fun="mean", ..., w=NULL, wide=TRUE,
as.raster=FALSE, filename="", overwrite=FALSE, wopt=list())

## S4 method for signature 'SpatRaster,SpatVector'
zonal(x, z, fun=mean, ..., w=NULL, weights=FALSE,
exact=FALSE, touches=FALSE, as.raster=FALSE, filename="", wopt=list())

## S4 method for signature 'SpatVector,SpatVector'
zonal(x, z, fun=mean, ..., weighted=FALSE, as.polygons=FALSE)
```

Arguments

<code>x</code>	<code>SpatRaster</code> or <code>SpatVector</code>
<code>z</code>	<code>SpatRaster</code> with cell-values representing zones or a <code>SpatVector</code> with each polygon geometry representing a zone. <code>z</code> can have multiple layers to define intersecting zones
<code>fun</code>	function to be applied to summarize the values by zone. Either as character: "mean", "min", "max", "sum", "isNA", and "notNA" and, for relatively small <code>SpatRasters</code> , a proper function
<code>...</code>	additional arguments passed to <code>fun</code> , such as <code>na.rm=TRUE</code>
<code>w</code>	<code>SpatRaster</code> with weights. Should have a single-layer with non-negative values
<code>wide</code>	logical. Should the values returned in a wide format? This only affects the results when <code>nlyr(z) == 2</code>
<code>as.raster</code>	logical. If TRUE, a <code>SpatRaster</code> is returned with the zonal statistic for each zone
<code>filename</code>	character. Output filename (ignored if <code>as.raster=FALSE</code>)
<code>overwrite</code>	logical. If TRUE, <code>filename</code> is overwritten
<code>wopt</code>	list with additional arguments for writing files as in writeRaster
<code>weights</code>	logical. If TRUE and <code>y</code> has polygons, the approximate fraction of each cell that is covered is returned as well, for example to compute a weighted mean

<code>exact</code>	logical. If TRUE and <code>y</code> has polygons, the exact fraction of each cell that is covered is returned as well, for example to compute a weighted mean
<code>touches</code>	logical. If TRUE, values for all cells touched by lines or polygons are extracted, not just those on the line render path, or whose center point is within the polygon. Not relevant for points; and always considered TRUE when <code>weights=TRUE</code> or <code>exact=TRUE</code>
<code>weighted</code>	logical. If TRUE, a <code>weighted.mean</code> is computed and <code>fun</code> is ignored. Weights are based on the length of the lines or the area of the polygons in <code>x</code> that intersect with <code>z</code> . This argument is ignored if <code>x</code> is a <code>SpatVector</code> or points
<code>as.polygons</code>	logical. Should the results be merged with the attributes of <code>z</code> ?

Value

A data.frame with a value for each zone, or a `SpatRaster`, or `SpatVector` of polygons.

See Also

See [global](#) for "global" statistics (i.e., all of `x` is considered a single zone), [app](#) for local statistics, and [extract](#) for an alternative way to summarize values of a `SpatRaster` with a `SpatVector`.

Examples

```
### SpatRaster, SpatRaster
r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)
z <- rast(r)
values(z) <- rep(c(1:2, NA, 3:4), each=20)
names(z) <- "zone"
zonal(r, z, "sum", na.rm=TRUE)

# with weights
w <- init(r, "col")
zonal(r, z, w=w, "mean", na.rm=TRUE)

# multiple layers
r <- rast(system.file("ex/logo.tif", package = "terra"))
# zonal layer
z <- rast(r, 1)
names(z) <- "zone"
values(z) <- rep(c(1:2, NA, c(3:4)), each=ncell(r)/5, length.out=ncell(r))

zonal(r, z, "mean", na.rm = TRUE)

# raster of zonal values
zr <- zonal(r, z, "mean", na.rm = TRUE, as.raster=TRUE)

### SpatRaster, SpatVector
x <- rast(ncol=2, nrow=2, vals=1:4, xmin=0, xmax=1, ymin=0, ymax=1, crs="+proj=utm +zone=1")
p <- as.polygons(x)
```

```

pp <- shift(p, .2)
r <- disagg(x, 4)

zonal(r, p)
zonal(r, p, sum)
zonal(x, pp, exact=TRUE)
zonal(c(x, x*10), pp, w=x)

### SpatVector, SpatVector

f <- system.file("ex/lux.shp", package="terra")
v <- vect(f)[,c(2,4)]

p <- spatSample(v, 100)
values(p) <- data.frame(b2=1:100, ssep1=100:1)

zonal(p, v, mean)

```

zoom

*Zoom in on a map***Description**

Zoom in on a map (plot) by providing a new extent, by default this is done by clicking twice on the map.

Usage

```

## S4 method for signature 'SpatRaster'
zoom(x, e=draw(), maxcell=100000, layer=1, new=FALSE, ...)

## S4 method for signature 'SpatVector'
zoom(x, e=draw(), new=FALSE, ...)

```

Arguments

x	SpatRaster
e	SpatExtent
maxcell	positive integer. Maximum number of cells used for the map
layer	positive integer to select the layer to be used
new	logical. If TRUE, the zoomed in map will appear on a new device (window)
...	additional arguments passed to plot

Value

SpatExtent (invisibly)

300

zoom

See Also

[draw, plot](#)

Index

- !, SpatRaster-method (Compare-methods), 61
- * **classes**
 - options, 169
 - SpatExtent-class, 240
 - SpatRaster-class, 241
 - SpatVector-class, 243
- * **math**
 - Arith-methods, 32
 - atan2, 38
 - Compare-methods, 61
 - Math-methods, 154
 - modal, 159
- * **methods**
 - activeCat, 20
 - add_legend, 22
 - aggregate, 24
 - animate, 28
 - app, 29
 - Arith-methods, 32
 - as.data.frame, 34
 - as.list, 35
 - as.raster, 36
 - barplot, 41
 - boundaries, 42
 - cartogram, 46
 - catalyze, 47
 - cells, 48
 - cellSize, 49
 - colors, 58
 - combineGeoms, 59
 - Compare-methods, 61
 - concat, 63
 - contour, 64
 - convHull, 65
 - cover, 67
 - crosstab, 71
 - deepcopy, 74
 - densify, 75
 - diff, 79
 - disagg, 83
 - dots, 87
 - elongate, 89
 - erase, 90
 - expanse, 91
 - extract, 95
 - extremes, 98
 - factors, 99
 - fillHoles, 101
 - fillTime, 102
 - focalValues, 113
 - gaps, 114
 - geomtype, 117
 - graticule, 119
 - headtail, 122
 - hist, 123
 - image, 125
 - impose, 126
 - inset, 130
 - interpolation, 134
 - intersect, 137
 - is.bool, 139
 - lapp, 141
 - lines, 146
 - makeTiles, 148
 - makeVRT, 149
 - mask, 151
 - match, 153
 - Math-methods, 154
 - merge, 156
 - mergeTime, 157
 - meta, 158
 - mosaic, 160
 - normalize.longitude, 166
 - not.na, 168
 - panel, 172
 - patches, 173
 - perim, 174

- persp, 175
- plet, 176
- plot, 178
- plot_extent, 184
- plot_graticule, 185
- plotRGB, 182
- predict, 186
- quantile, 192
- query, 193
- rapp, 195
- rast, 197
- readwrite, 205
- relate, 208
- replace_layers, 212
- replace_values, 213
- RGB, 216
- sapp, 221
- scatterplot, 224
- scoff, 225
- sds, 226
- selectRange, 230
- serialize, 231
- setValues, 232
- sharedPaths, 234
- simplifyGeom, 237
- sources, 239
- split, 245
- sprc, 246
- summarize, 254
- summary, 257
- svc, 258
- syndif, 259
- tapp, 260
- text, 263
- topology, 267
- union, 270
- values, 275
- vect, 276
- vector_layers, 279
- vrt, 282
- width, 286
- window, 287
- wrap, 288
- writeCDF, 289
- writeRaster, 290
- writeVector, 292
- * **package**
 - terra-package, 7
- * **spatial**
 - activeCat, 20
 - add, 21
 - add_legend, 22
 - adjacent, 22
 - aggregate, 24
 - align, 25
 - all.equal, 27
 - animate, 28
 - app, 29
 - approximate, 31
 - Arith-methods, 32
 - as.character, 33
 - as.data.frame, 34
 - as.list, 35
 - as.raster, 36
 - as.spatvector, 37
 - atan2, 38
 - autocorrelation, 39
 - barplot, 41
 - boundaries, 42
 - boxplot, 43
 - buffer, 44
 - c, 45
 - cartogram, 46
 - catalyze, 47
 - cells, 48
 - cellSize, 49
 - centroids, 51
 - clamp, 52
 - clamp_ts, 53
 - classify, 54
 - click, 56
 - coerce, 57
 - colors, 58
 - combineGeoms, 59
 - Compare-methods, 61
 - compareGeom, 62
 - concat, 63
 - contour, 64
 - convHull, 65
 - costDist, 66
 - cover, 67
 - crds, 68
 - crop, 69
 - crosstab, 71
 - crs, 72
 - deepcopy, 74

densify, 75
density, 76
depth, 77
describe, 78
diff, 79
dimensions, 80
direction, 82
disagg, 83
distance, 84
dots, 87
draw, 88
elongate, 89
erase, 90
expanse, 91
ext, 93
extend, 94
extract, 95
extremes, 98
factors, 99
fillHoles, 101
fillTime, 102
flip, 103
focal, 104
focal3D, 106
focalCpp, 107
focalMat, 109
focalPairs, 110
focalReg, 112
focalValues, 113
freq, 113
gaps, 114
gdal, 115
geom, 116
geomtype, 117
global, 118
graticule, 119
gridDistance, 120
headtail, 122
hist, 123
ifel, 124
image, 125
impose, 126
initialize, 127
inplace, 128
inset, 130
interpIDW, 131
interpNear, 133
interpolation, 134
intersect, 137
is.bool, 139
is.lonlat, 140
lapp, 141
linearUnits, 145
lines, 146
makeTiles, 148
makeVRT, 149
map.pal, 150
mask, 151
match, 153
Math-methods, 154
mem, 155
merge, 156
mergeTime, 157
meta, 158
mosaic, 160
na.omit, 161
NAflag, 162
names, 163
nearest, 165
normalize.longitude, 166
north, 166
not.na, 168
options, 169
origin, 170
pairs, 171
panel, 172
patches, 173
perim, 174
persp, 175
plet, 176
plot, 178
plot_extent, 184
plot_graticule, 185
plotRGB, 182
predict, 186
project, 189
quantile, 192
query, 193
rangeFill, 194
rapp, 195
rast, 197
rasterize, 200
rasterizeGeom, 201
rasterizeWin, 203
rcl, 204
readwrite, 205

- rectify, [207](#)
- relate, [208](#)
- rep, [210](#)
- replace_dollar, [211](#)
- replace_layers, [212](#)
- replace_values, [213](#)
- resample, [214](#)
- rescale, [215](#)
- RGB, [216](#)
- roll, [217](#)
- rotate, [219](#)
- same.crs, [220](#)
- sapp, [221](#)
- sbar, [222](#)
- scale, [223](#)
- scatterplot, [224](#)
- scoff, [225](#)
- sds, [226](#)
- segregate, [227](#)
- sel, [228](#)
- selectHighest, [229](#)
- selectRange, [230](#)
- serialize, [231](#)
- setValues, [232](#)
- shade, [233](#)
- sharedPaths, [234](#)
- shift, [235](#)
- sieve, [236](#)
- simplifyGeom, [237](#)
- sort, [238](#)
- sources, [239](#)
- SpatExtent-class, [240](#)
- SpatRaster-class, [241](#)
- spatSample, [241](#)
- SpatVector-class, [243](#)
- spin, [244](#)
- split, [245](#)
- sprc, [246](#)
- stretch, [247](#)
- subset, [248](#)
- subset_dollar, [249](#)
- subset_double, [251](#)
- subset_single, [252](#)
- subst, [253](#)
- summarize, [254](#)
- summary, [257](#)
- svc, [258](#)
- symdif, [259](#)
- tapp, [260](#)
- terra-package, [7](#)
- terrain, [261](#)
- text, [263](#)
- tighten, [264](#)
- time, [265](#)
- tmpFiles, [266](#)
- topology, [267](#)
- transpose, [268](#)
- trim, [269](#)
- union, [270](#)
- unique, [271](#)
- units, [272](#)
- update, [273](#)
- valid, [274](#)
- values, [275](#)
- vect, [276](#)
- vector_layers, [279](#)
- viewshed, [280](#)
- voronoi, [281](#)
- vrt, [282](#)
- where, [284](#)
- which.lyr, [285](#)
- width, [286](#)
- window, [287](#)
- wrap, [288](#)
- writeCDF, [289](#)
- writeRaster, [290](#)
- writeVector, [292](#)
- xmin, [293](#)
- xyRowColCell, [294](#)
- zonal, [297](#)
- zoom, [299](#)
- * **univar**
 - freq, [113](#)
 - modal, [159](#)
- [, [13](#), [16](#), [138](#)
- [(subset_single), [252](#)
- [(SpatExtent,missing,missing-method (subset_single), [252](#)
- [(SpatExtent,numeric,missing-method (subset_single), [252](#)
- [(SpatRaster,ANY,ANY,ANY-method (subset_single), [252](#)
- [(SpatRaster,ANY,ANY-method (subset_single), [252](#)
- [(SpatRaster,SpatExtent,missing-method (subset_single), [252](#)

- [, SpatRaster, SpatRaster, missing-method (subset_single), [252](#)
- [, SpatRaster, SpatVector, missing-method (subset_single), [252](#)
- [, SpatRaster, data.frame, missing-method (subset_single), [252](#)
- [, SpatRaster, matrix, missing-method (subset_single), [252](#)
- [, SpatRaster, missing, missing-method (subset_single), [252](#)
- [, SpatRaster, missing, numeric-method (subset_single), [252](#)
- [, SpatRaster, numeric, missing-method (subset_single), [252](#)
- [, SpatRaster, numeric, numeric-method (subset_single), [252](#)
- [, SpatRasterCollection, numeric, missing-method (subset_single), [252](#)
- [, SpatRasterDataset, character, missing-method (subset_single), [252](#)
- [, SpatRasterDataset, logical, missing-method (subset_single), [252](#)
- [, SpatRasterDataset, missing, logical-method (subset_single), [252](#)
- [, SpatRasterDataset, missing, numeric-method (subset_single), [252](#)
- [, SpatRasterDataset, numeric, logical-method (subset_single), [252](#)
- [, SpatRasterDataset, numeric, missing-method (subset_single), [252](#)
- [, SpatRasterDataset, numeric, numeric-method (subset_single), [252](#)
- [, SpatVector, SpatExtent, missing-method (subset_single), [252](#)
- [, SpatVector, SpatVector, missing-method (subset_single), [252](#)
- [, SpatVector, character, missing-method (subset_single), [252](#)
- [, SpatVector, data.frame, ANY-method (subset_single), [252](#)
- [, SpatVector, data.frame, missing-method (subset_single), [252](#)
- [, SpatVector, logical, character-method (subset_single), [252](#)
- [, SpatVector, logical, logical-method (subset_single), [252](#)
- [, SpatVector, logical, missing-method (subset_single), [252](#)
- [, SpatVector, logical, numeric-method (subset_single), [252](#)
- [, SpatVector, matrix, missing-method (subset_single), [252](#)
- [, SpatVector, missing, character-method (subset_single), [252](#)
- [, SpatVector, missing, logical-method (subset_single), [252](#)
- [, SpatVector, missing, missing-method (subset_single), [252](#)
- [, SpatVector, missing, numeric-method (subset_single), [252](#)
- [, SpatVector, numeric, character-method (subset_single), [252](#)
- [, SpatVector, numeric, logical-method (subset_single), [252](#)
- [, SpatVector, numeric, missing-method (subset_single), [252](#)
- [, SpatVector, numeric, numeric-method (subset_single), [252](#)
- [, SpatVectorCollection, numeric, missing-method (svc), [258](#)
- [<- (replace_values), [213](#)
- [<-, SpatExtent, numeric, missing-method (replace_values), [213](#)
- [<-, SpatRaster, ANY, ANY, ANY-method (replace_values), [213](#)
- [<-, SpatRaster, ANY, ANY-method (replace_values), [213](#)
- [<-, SpatRasterDataset, numeric, missing-method (sds), [226](#)
- [<-, SpatVector, ANY, ANY-method (replace_values), [213](#)
- [<-, SpatVector, ANY, missing-method (replace_values), [213](#)
- [<-, SpatVector, missing, ANY-method (replace_values), [213](#)
- [<-, SpatVectorCollection, numeric, missing-method (svc), [258](#)
- [[(subset_double), [251](#)
- [[, SpatRaster, ANY, missing-method (subset_double), [251](#)
- [[, SpatRaster, character, missing-method (subset_double), [251](#)
- [[, SpatRaster, logical, missing-method (subset_double), [251](#)
- [[, SpatRaster, numeric, missing-method (subset_double), [251](#)

- [[, SpatRasterDataset, ANY, ANY-method
(subset_double), [251](#)
- [[, SpatVector, character, missing-method
(subset_double), [251](#)
- [[, SpatVector, logical, missing-method
(subset_double), [251](#)
- [[, SpatVector, numeric, missing-method
(subset_double), [251](#)
- [[, SpatVectorCollection, numeric, missing-method
(svc), [258](#)
- [[<- (replace_layers), [212](#)
- [[<-, SpatRaster, character-method
(replace_layers), [212](#)
- [[<-, SpatRaster, numeric-method
(replace_layers), [212](#)
- [[<-, SpatVector, character-method
(replace_layers), [212](#)
- [[<-, SpatVector, numeric-method
(replace_layers), [212](#)
- \$, [211](#)
- \$(subset_dollar), [249](#)
- \$(SpatExtent-method (subset_dollar), [249](#)
- \$(SpatRaster-method (subset_dollar), [249](#)
- \$(SpatRasterDataset-method
(subset_dollar), [249](#)
- \$(SpatVector-method (subset_dollar), [249](#)
- \$<- (replace_dollar), [211](#)
- \$<-, SpatExtent-method (replace_dollar),
[211](#)
- \$<-, SpatRaster-method (replace_dollar),
[211](#)
- \$<-, SpatVector-method (replace_dollar),
[211](#)
- %in% (match), [153](#)
- %in%, SpatRaster-method (match), [153](#)
- activeCat, [12](#), [20](#), [48](#), [100](#)
- activeCat, SpatRaster-method
(activeCat), [20](#)
- activeCat<- (activeCat), [20](#)
- activeCat<-, SpatRaster-method
(activeCat), [20](#)
- add, [21](#)
- add<-, [7](#), [18](#)
- add<- (add), [21](#)
- add<-, SpatRaster, SpatRaster-method
(add), [21](#)
- add_legend, [22](#)
- adjacent, [9](#), [14](#), [22](#), [165](#), [208](#)
- adjacent, SpatRaster-method (adjacent),
[22](#)
- adjacent, SpatVector-method (adjacent),
[22](#)
- aggregate, [8](#), [14](#), [24](#), [83](#), [214](#), [215](#), [260](#), [270](#)
- aggregate, SpatRaster-method
(aggregate), [24](#)
- aggregate, SpatVector-method
(aggregate), [24](#)
- align, [16](#), [25](#)
- align, SpatExtent, numeric-method
(align), [25](#)
- align, SpatExtent, SpatRaster-method
(align), [25](#)
- all (summarize), [254](#)
- all, SpatRaster-method (summarize), [254](#)
- all.equal, [27](#), [62](#)
- all.equal, SpatRaster, SpatRaster-method
(all.equal), [27](#)
- allNA (summarize), [254](#)
- allNA, SpatRaster-method (summarize), [254](#)
- animate, [28](#)
- animate, SpatRaster-method (animate), [28](#)
- any (summarize), [254](#)
- any, SpatRaster-method (summarize), [254](#)
- anyNA (summarize), [254](#)
- anyNA, SpatRaster-method (summarize), [254](#)
- app, [8](#), [18](#), [29](#), [33](#), [62](#), [119](#), [142](#), [153](#), [155](#), [193](#),
[195](#), [196](#), [221](#), [254](#), [256](#), [260](#), [261](#),
[298](#)
- app, SpatRaster-method (app), [29](#)
- app, SpatRasterDataset-method (app), [29](#)
- apply, [29](#)
- approx, [31](#)
- approximate, [9](#), [18](#), [31](#), [102](#)
- approximate, SpatRaster-method
(approximate), [31](#)
- area (deprecated), [77](#)
- area, SpatRaster-method (deprecated), [77](#)
- Arith, logical, SpatRaster-method
(Arith-methods), [32](#)
- Arith, matrix, SpatRaster-method
(Arith-methods), [32](#)
- Arith, missing, SpatRaster-method
(Arith-methods), [32](#)
- Arith, numeric, SpatExtent-method
(Arith-methods), [32](#)
- Arith, numeric, SpatRaster-method

- (Arith-methods), [32](#)
- Arith, SpatExtent, numeric-method (Arith-methods), [32](#)
- Arith, SpatExtent, SpatExtent-method (Arith-methods), [32](#)
- Arith, SpatRaster, logical-method (Arith-methods), [32](#)
- Arith, SpatRaster, matrix-method (Arith-methods), [32](#)
- Arith, SpatRaster, missing-method (Arith-methods), [32](#)
- Arith, SpatRaster, numeric-method (Arith-methods), [32](#)
- Arith, SpatRaster, SpatRaster-method (Arith-methods), [32](#)
- Arith, SpatVector, SpatVector-method (Arith-methods), [32](#)
- Arith-methods, [8](#), [32](#), [62](#)
- arrows, [147](#)
- as.array, [10](#)
- as.array (coerce), [57](#)
- as.array, SpatRaster-method (coerce), [57](#)
- as.bool, [8](#)
- as.bool (is.bool), [139](#)
- as.bool, SpatRaster-method (is.bool), [139](#)
- as.character, [33](#)
- as.character, SpatExtent-method (as.character), [33](#)
- as.character, SpatRaster-method (as.character), [33](#)
- as.contour, [17](#)
- as.contour (contour), [64](#)
- as.contour, SpatRaster-method (contour), [64](#)
- as.data.frame, [15](#), [34](#), [58](#), [275](#)
- as.data.frame, SpatRaster-method (as.data.frame), [34](#)
- as.data.frame, SpatVector-method (as.data.frame), [34](#)
- as.factor, [12](#), [100](#)
- as.factor (is.bool), [139](#)
- as.factor, SpatRaster-method (is.bool), [139](#)
- as.int, [8](#)
- as.int (is.bool), [139](#)
- as.int, SpatRaster-method (is.bool), [139](#)
- as.integer, SpatRaster-method (is.bool), [139](#)
- as.lines, [17](#)
- as.lines (as.spatvector), [37](#)
- as.lines, SpatExtent-method (as.spatvector), [37](#)
- as.lines, SpatRaster-method (as.spatvector), [37](#)
- as.lines, SpatVector-method (as.spatvector), [37](#)
- as.list, [15](#), [19](#), [35](#), [35](#)
- as.list, SpatRaster-method (as.list), [35](#)
- as.list, SpatRasterCollection-method (as.list), [35](#)
- as.list, SpatRasterDataset-method (as.list), [35](#)
- as.list, SpatVector-method (as.list), [35](#)
- as.list, SpatVectorCollection-method (as.list), [35](#)
- as.logical, SpatRaster-method (is.bool), [139](#)
- as.matrix, [10](#), [35](#), [276](#)
- as.matrix (coerce), [57](#)
- as.matrix, SpatRaster-method (coerce), [57](#)
- as.numeric, [12](#)
- as.numeric (catalyze), [47](#)
- as.numeric, SpatRaster-method (catalyze), [47](#)
- as.points, [17](#), [19](#)
- as.points (as.spatvector), [37](#)
- as.points, SpatExtent-method (as.spatvector), [37](#)
- as.points, SpatRaster-method (as.spatvector), [37](#)
- as.points, SpatVector-method (as.spatvector), [37](#)
- as.polygons, [17](#), [19](#), [58](#)
- as.polygons (as.spatvector), [37](#)
- as.polygons, SpatExtent-method (as.spatvector), [37](#)
- as.polygons, SpatRaster-method (as.spatvector), [37](#)
- as.polygons, SpatVector-method (as.spatvector), [37](#)
- as.raster, [36](#), [36](#)
- as.raster, SpatRaster-method (as.raster), [36](#)
- as.spatvector, [37](#)
- as.vector (coerce), [57](#)
- as.vector, SpatRaster-method (coerce), [57](#)

- atan2, [38](#)
- atan2, SpatRaster, SpatRaster-method (atan2), [38](#)
- atan_2 (atan2), [38](#)
- atan_2, SpatRaster, SpatRaster-method (atan2), [38](#)
- autocor, [10](#)
- autocor (autocorrelation), [39](#)
- autocor, numeric-method (autocorrelation), [39](#)
- autocor, SpatRaster-method (autocorrelation), [39](#)
- autocorrelation, [39](#)
- axis, [179](#)
- barplot, [18](#), [41](#), [41](#)
- barplot, SpatRaster-method (barplot), [41](#)
- blocks, [12](#)
- blocks (readwrite), [205](#)
- blocks, SpatRaster-method (readwrite), [205](#)
- boundaries, [9](#), [42](#), [173](#)
- boundaries, SpatRaster-method (boundaries), [42](#)
- boxplot, [18](#), [41](#), [43](#), [43](#), [123](#), [171](#)
- boxplot, SpatRaster-method (boxplot), [43](#)
- buffer, [14](#), [44](#), [89](#), [137](#)
- buffer, SpatRaster-method (buffer), [44](#)
- buffer, SpatVector-method (buffer), [44](#)
- bxp, [43](#)
- c, [7](#), [18](#), [21](#), [45](#), [157](#), [270](#)
- c, SpatRaster-method (c), [45](#)
- c, SpatRasterDataset-method (c), [45](#)
- c, SpatVector-method (c), [45](#)
- c, SpatVectorCollection-method (c), [45](#)
- cartogram, [17](#), [46](#), [87](#)
- cartogram, SpatVector-method (cartogram), [46](#)
- catalyze, [12](#), [47](#), [100](#)
- catalyze, SpatRaster-method (catalyze), [47](#)
- categories, [128](#)
- categories (factors), [99](#)
- categories, SpatRaster-method (factors), [99](#)
- cats, [12](#), [20](#), [48](#), [64](#), [139](#), [140](#)
- cats (factors), [99](#)
- cats, SpatRaster-method (factors), [99](#)
- cellFromRowCol, [11](#), [205](#)
- cellFromRowCol (xyRowColCell), [294](#)
- cellFromRowCol, SpatRaster, numeric, numeric-method (xyRowColCell), [294](#)
- cellFromRowColCombine, [11](#)
- cellFromRowColCombine (xyRowColCell), [294](#)
- cellFromRowColCombine, SpatRaster, numeric, numeric-method (xyRowColCell), [294](#)
- cellFromXY, [11](#)
- cellFromXY (xyRowColCell), [294](#)
- cellFromXY, SpatRaster, data.frame-method (xyRowColCell), [294](#)
- cellFromXY, SpatRaster, matrix-method (xyRowColCell), [294](#)
- cells, [11](#), [18](#), [48](#), [96](#)
- cells, SpatRaster, missing-method (cells), [48](#)
- cells, SpatRaster, numeric-method (cells), [48](#)
- cells, SpatRaster, SpatExtent-method (cells), [48](#)
- cells, SpatRaster, SpatVector-method (cells), [48](#)
- cellSize, [9](#), [18](#), [49](#), [92](#)
- cellSize, SpatRaster-method (cellSize), [49](#)
- centroids, [13](#), [51](#)
- centroids, SpatVector-method (centroids), [51](#)
- clamp, [52](#), [53](#)
- clamp, numeric-method (clamp), [52](#)
- clamp, SpatRaster-method (clamp), [52](#)
- clamp_ts, [53](#)
- clamp_ts, numeric-method (clamp_ts), [53](#)
- clamp_ts, SpatRaster-method (clamp_ts), [53](#)
- classify, [9](#), [19](#), [52](#), [54](#), [124](#), [157](#), [162](#), [213](#), [253](#), [254](#)
- classify, SpatRaster-method (classify), [54](#)
- clearance, [15](#)
- clearance (width), [286](#)
- clearance, SpatVector-method (width), [286](#)
- click, [15](#), [18](#), [56](#), [88](#), [228](#)
- click, missing-method (click), [56](#)
- click, SpatRaster-method (click), [56](#)
- click, SpatVector-method (click), [56](#)

- coerce, [36](#), [57](#)
- colFromCell (xyRowColCell), [294](#)
- colFromCell, SpatRaster, numeric-method (xyRowColCell), [294](#)
- colFromX, [11](#)
- colFromX (xyRowColCell), [294](#)
- colFromX, SpatRaster, numeric-method (xyRowColCell), [294](#)
- colorize, [17](#), [183](#), [184](#)
- colorize (RGB), [216](#)
- colorize, SpatRaster-method (RGB), [216](#)
- colorRamp, [150](#)
- colors, [58](#)
- coltab (colors), [58](#)
- coltab, SpatRaster-method (colors), [58](#)
- coltab<- (colors), [58](#)
- coltab<-, SpatRaster-method (colors), [58](#)
- combineGeoms, [16](#), [59](#)
- combineGeoms, SpatVector, SpatVector-method (combineGeoms), [59](#)
- compare (Compare-methods), [61](#)
- Compare, matrix, SpatRaster-method (Compare-methods), [61](#)
- Compare, numeric, SpatRaster-method (Compare-methods), [61](#)
- Compare, SpatExtent, SpatExtent-method (Compare-methods), [61](#)
- Compare, SpatRaster, character-method (Compare-methods), [61](#)
- Compare, SpatRaster, matrix-method (Compare-methods), [61](#)
- Compare, SpatRaster, numeric-method (Compare-methods), [61](#)
- Compare, SpatRaster, SpatRaster-method (Compare-methods), [61](#)
- compare, SpatRaster-method (Compare-methods), [61](#)
- Compare-methods, [8](#), [61](#)
- compareGeom, [11](#), [18](#), [27](#), [62](#)
- compareGeom, SpatRaster, SpatRaster-method (compareGeom), [62](#)
- compareGeom, SpatVector, missing-method (compareGeom), [62](#)
- compareGeom, SpatVector, SpatVector-method (compareGeom), [62](#)
- concats, [12](#), [63](#)
- concats, SpatRaster-method (concats), [63](#)
- contour, [17](#), [64](#), [64](#)
- contour, SpatRaster-method (contour), [64](#)
- convHull, [14](#), [65](#)
- convHull, SpatVector-method (convHull), [65](#)
- cor, [171](#)
- costDist, [9](#), [66](#), [121](#)
- costDist, SpatRaster-method (costDist), [66](#)
- costDistance (costDist), [66](#)
- costDistance, SpatRaster-method (costDist), [66](#)
- countNA (summarize), [254](#)
- countNA, SpatRaster-method (summarize), [254](#)
- cov.wt, [145](#)
- cover, [9](#), [14](#), [67](#), [124](#)
- cover, SpatRaster, SpatRaster-method (cover), [67](#)
- cover, SpatVector, SpatVector-method (cover), [67](#)
- cppFunction, [108](#)
- crds, [14](#), [18](#), [68](#), [117](#), [296](#)
- crds, SpatRaster-method (crds), [68](#)
- crds, SpatVector-method (crds), [68](#)
- crop, [8](#), [13](#), [15](#), [16](#), [69](#), [89](#), [90](#), [94](#), [95](#), [138](#), [152](#), [208](#), [214](#), [215](#), [228](#), [270](#), [287](#)
- crop, SpatGraticule-method (crop), [69](#)
- crop, SpatRaster-method (crop), [69](#)
- crop, SpatRasterCollection-method (crop), [69](#)
- crop, SpatRasterDataset-method (crop), [69](#)
- crop, SpatVector-method (crop), [69](#)
- crosstab, [9](#), [71](#)
- crosstab, SpatRaster, missing-method (crosstab), [71](#)
- crs, [11](#), [14](#), [38](#), [72](#), [145](#), [190](#), [191](#), [198](#), [242](#), [277](#)
- crs, character-method (crs), [72](#)
- crs, sf-method (crs), [72](#)
- crs, SpatExtent-method (crs), [72](#)
- crs, SpatRaster-method (crs), [72](#)
- crs, SpatRasterDataset-method (crs), [72](#)
- crs, SpatVector-method (crs), [72](#)
- crs, SpatVectorProxy-method (crs), [72](#)
- crs<- (crs), [72](#)
- crs<-, SpatRaster, ANY-method (crs), [72](#)
- crs<-, SpatRaster-method (crs), [72](#)
- crs<-, SpatVector, ANY-method (crs), [72](#)

- crs<-, SpatVector-method (crs), 72
- cummax, 53
- cummin, 53
- cumsum, 218
- cumsum (Math-methods), 154
- cumsum, SpatRaster-method
(Math-methods), 154
- cut, 41
- data.frame, 35, 207, 275
- datatype (geomtype), 117
- datatype, SpatRaster-method (geomtype),
117
- datatype, SpatVector-method (geomtype),
117
- Date, 265
- deepcopy, 74
- deepcopy, SpatRaster-method (deepcopy),
74
- deepcopy, SpatVector-method (deepcopy),
74
- delaunay, 14
- delaunay (voronoi), 281
- delaunay, SpatVector-method (voronoi),
281
- deldir, 281
- densify, 75
- densify, SpatVector-method (densify), 75
- density, 18, 76
- density, SpatRaster-method (density), 76
- deprecated, 77
- depth, 77, 266
- depth, SpatRaster-method (depth), 77
- depth<- (depth), 77
- depth<-, SpatRaster-method (depth), 77
- describe, 78, 116, 226, 227
- describe, character-method (describe), 78
- diff, 79
- diff, SpatRaster-method (diff), 79
- dim (dimensions), 80
- dim, SpatRaster-method (dimensions), 80
- dim, SpatRasterCollection-method
(dimensions), 80
- dim, SpatRasterDataset-method
(dimensions), 80
- dim, SpatVector-method (dimensions), 80
- dim, SpatVectorProxy-method
(dimensions), 80
- dim<-, SpatRaster-method (dimensions), 80
- dimensions, 80
- direction, 9, 82
- direction, SpatRaster-method
(direction), 82
- disagg, 8, 15, 18, 25, 69, 83, 214, 215
- disagg, SpatRaster-method (disagg), 83
- disagg, SpatVector-method (disagg), 83
- distance, 9, 18, 45, 66, 82, 84, 121
- distance, matrix, matrix-method
(distance), 84
- distance, matrix, missing-method
(distance), 84
- distance, SpatRaster, missing-method
(distance), 84
- distance, SpatRaster, sf-method
(distance), 84
- distance, SpatRaster, SpatVector-method
(distance), 84
- distance, SpatVector, ANY-method
(distance), 84
- distance, SpatVector, SpatVector-method
(distance), 84
- dots, 17, 87
- dots, SpatVector-method (dots), 87
- draw, 16, 18, 19, 26, 57, 88, 183, 228, 300
- draw, character-method (draw), 88
- draw, missing-method (draw), 88
- droplevels (factors), 99
- droplevels, SpatRaster-method (factors),
99
- elongate, 16, 45, 89
- elongate, SpatVector-method (elongate),
89
- emptyGeoms (topology), 267
- emptyGeoms, SpatVector-method
(topology), 267
- erase, 14, 16, 60, 89, 90, 259
- erase, SpatGraticule, SpatVector-method
(erase), 90
- erase, SpatVector, missing-method
(erase), 90
- erase, SpatVector, SpatExtent-method
(erase), 90
- erase, SpatVector, SpatVector-method
(erase), 90
- expanse, 9, 14, 18, 50, 91
- expanse, SpatRaster-method (expanse), 91
- expanse, SpatVector-method (expanse), 91

- ext, [11](#), [14](#), [16](#), [19](#), [26](#), [70](#), [81](#), [93](#), [95](#), [240](#)
- ext, bbox-method (ext), [93](#)
- ext, Extent-method (ext), [93](#)
- ext, matrix-method (ext), [93](#)
- ext, missing-method (ext), [93](#)
- ext, numeric-method (ext), [93](#)
- ext, Raster-method (ext), [93](#)
- ext, sf-method (ext), [93](#)
- ext, SpatExtent-method (ext), [93](#)
- ext, Spatial-method (ext), [93](#)
- ext, SpatRaster-method (ext), [93](#)
- ext, SpatRasterCollection-method (ext), [93](#)
- ext, SpatRasterDataset-method (ext), [93](#)
- ext, SpatVector-method (ext), [93](#)
- ext, SpatVectorProxy-method (ext), [93](#)
- ext<- (ext), [93](#)
- ext<-, SpatRaster, numeric-method (ext), [93](#)
- ext<-, SpatRaster, SpatExtent-method (ext), [93](#)
- extend, [8](#), [69](#), [70](#), [94](#), [214](#), [270](#), [287](#)
- extend, SpatExtent-method (extend), [94](#)
- extend, SpatRaster-method (extend), [94](#)
- extract, [10](#), [15](#), [19](#), [95](#), [119](#), [230](#), [298](#)
- extract, SpatRaster, data.frame-method (extract), [95](#)
- extract, SpatRaster, matrix-method (extract), [95](#)
- extract, SpatRaster, numeric-method (extract), [95](#)
- extract, SpatRaster, sf-method (extract), [95](#)
- extract, SpatRaster, SpatExtent-method (extract), [95](#)
- extract, SpatRaster, SpatVector-method (extract), [95](#)
- extract, SpatVector, data.frame-method (extract), [95](#)
- extract, SpatVector, matrix-method (extract), [95](#)
- extract, SpatVector, SpatVector-method (extract), [95](#)
- extremes, [98](#)
- factor, [99](#)
- factors, [99](#)
- fileBlocksize (readwrite), [205](#)
- filled.contour, [64](#)
- fillHoles, [14](#), [15](#), [101](#), [115](#)
- fillHoles, SpatVector-method (fillHoles), [101](#)
- fillTime, [11](#), [32](#), [102](#), [157](#)
- fillTime, SpatRaster-method (fillTime), [102](#)
- flip, [8](#), [15](#), [103](#), [216](#), [236](#), [269](#)
- flip, SpatRaster-method (flip), [103](#)
- flip, SpatVector-method (flip), [103](#)
- focal, [9](#), [31](#), [32](#), [42](#), [104](#), [106–112](#), [119](#), [173](#), [237](#), [262](#)
- focal, SpatRaster-method (focal), [104](#)
- focal3D, [9](#), [105](#), [106](#), [111](#), [112](#)
- focal3D, SpatRaster-method (focal3D), [106](#)
- focalCor, [18](#), [105](#)
- focalCor (focalPairs), [110](#)
- focalCor, SpatRaster-method (focalPairs), [110](#)
- focalCpp, [9](#), [105](#), [107](#)
- focalCpp, SpatRaster-method (focalCpp), [107](#)
- focalMat, [105](#), [109](#)
- focalPairs, [9](#), [110](#)
- focalPairs, SpatRaster-method (focalPairs), [110](#)
- focalReg, [9](#), [105](#), [111](#), [112](#)
- focalReg, SpatRaster-method (focalReg), [112](#)
- focalValues, [105](#), [108](#), [112](#), [113](#), [276](#)
- focalValues, SpatRaster-method (focalValues), [113](#)
- free_RAM (mem), [155](#)
- freq, [9](#), [71](#), [113](#)
- freq, SpatRaster-method (freq), [113](#)
- gaps, [15](#), [114](#), [235](#), [237](#), [268](#)
- gaps, SpatVector, SpatExtent-method (gaps), [114](#)
- gaps, SpatVector-method (gaps), [114](#)
- gdal, [115](#), [199](#), [291](#)
- gdalCache (gdal), [115](#)
- geom, [14](#), [35](#), [36](#), [69](#), [116](#), [123](#), [276–278](#)
- geom, SpatVector-method (geom), [116](#)
- geomtype, [117](#)
- geomtype, Spatial-method (geomtype), [117](#)
- geomtype, SpatVector-method (geomtype), [117](#)
- geomtype, SpatVectorProxy-method (geomtype), [117](#)

- getGDALconfig (gdal), 115
- global, 9, 18, 92, 118, 145, 192, 257, 258, 298
- global, SpatRaster-method (global), 118
- graticule, 16, 119, 185, 186
- gridDist, 9, 66
- gridDist (gridDistance), 120
- gridDist, SpatRaster-method (gridDistance), 120
- gridDistance, 120
- gridDistance, SpatRaster-method (gridDistance), 120
- halo, 17, 121, 264
- has.colors (colors), 58
- has.colors, SpatRaster-method (colors), 58
- has.RGB (RGB), 216
- has.RGB, SpatRaster-method (RGB), 216
- hasMinMax (extremes), 98
- hasMinMax, SpatRaster-method (extremes), 98
- hasValues (sources), 239
- hasValues, SpatRaster-method (sources), 239
- head (headtail), 122
- head, SpatRaster-method (headtail), 122
- head, SpatVector-method (headtail), 122
- headtail, 122
- hist, 18, 41, 43, 123, 123, 171
- hist, SpatRaster-method (hist), 123
- ifel, 33, 62, 124
- ifel, SpatRaster-method (ifel), 124
- ifelse, 124
- image, 17, 125, 125, 181, 186
- image, SpatRaster-method (image), 125
- impose, 13, 126
- impose, SpatRasterCollection-method (impose), 126
- inext (inset), 130
- inext, SpatVector-method (inset), 130
- init, 9, 232, 233
- init (initialize), 127
- init, SpatRaster-method (initialize), 127
- initialize, 127
- inMemory, 11, 13
- inMemory (sources), 239
- inMemory, SpatRaster-method (sources), 239
- inplace, 128
- inset, 17, 130, 167, 215, 216, 222
- inset, SpatRaster-method (inset), 130
- inset, SpatVector-method (inset), 130
- interpIDW, 10, 131, 134, 204
- interpIDW, SpatRaster, matrix-method (interpIDW), 131
- interpIDW, SpatRaster, SpatVector-method (interpIDW), 131
- interpNear, 10, 132, 133, 204
- interpNear, SpatRaster, matrix-method (interpNear), 133
- interpNear, SpatRaster, SpatVector-method (interpNear), 133
- interpolate, 10, 132, 134, 187
- interpolate (interpolation), 134
- interpolate, SpatRaster-method (interpolation), 134
- interpolation, 134
- intersect, 14, 16, 60, 69, 70, 90, 137, 208, 228, 270
- intersect, SpatExtent, SpatExtent-method (intersect), 137
- intersect, SpatExtent, SpatVector-method (intersect), 137
- intersect, SpatRaster, SpatRaster-method (intersect), 137
- intersect, SpatVector, SpatExtent-method (intersect), 137
- intersect, SpatVector, SpatVector-method (intersect), 137
- is.bool, 139
- is.bool, SpatRaster-method (is.bool), 139
- is.factor, 12, 100
- is.factor (is.bool), 139
- is.factor, SpatRaster-method (is.bool), 139
- is.finite, SpatRaster-method (Compare-methods), 61
- is.infinite, SpatRaster-method (Compare-methods), 61
- is.int (is.bool), 139
- is.int, SpatRaster-method (is.bool), 139
- is.lines (geomtype), 117
- is.lines, SpatVector-method (geomtype), 117
- is.lonlat, 11, 14, 18, 19, 140
- is.lonlat, character-method (is.lonlat),

- 140
- is.lonlat, SpatRaster-method
(is.lonlat), 140
- is.lonlat, SpatVector-method
(is.lonlat), 140
- is.na, SpatRaster-method
(Compare-methods), 61
- is.nan, SpatRaster-method
(Compare-methods), 61
- is.points (geomtype), 117
- is.points, SpatVector-method (geomtype),
117
- is.polygons (geomtype), 117
- is.polygons, SpatVector-method
(geomtype), 117
- is.related, 138
- is.related (relate), 208
- is.related, SpatExtent, SpatRaster-method
(relate), 208
- is.related, SpatExtent, SpatVector-method
(relate), 208
- is.related, SpatRaster, SpatExtent-method
(relate), 208
- is.related, SpatRaster, SpatRaster-method
(relate), 208
- is.related, SpatRaster, SpatVector-method
(relate), 208
- is.related, SpatVector, SpatExtent-method
(relate), 208
- is.related, SpatVector, SpatRaster-method
(relate), 208
- is.related, SpatVector, SpatVector-method
(relate), 208
- is.valid, 15, 237
- is.valid (valid), 274
- is.valid, SpatExtent-method (valid), 274
- is.valid, SpatVector-method (valid), 274
- isFALSE, SpatRaster-method (is.bool), 139
- isTRUE, 285
- isTRUE, SpatRaster-method (is.bool), 139

- lapp, 8, 19, 30, 141, 196, 221
- lapp, SpatRaster-method (lapp), 141
- lapp, SpatRasterDataset-method (lapp),
141
- lapply, 221
- layerCor, 9, 19, 111, 144
- layerCor, SpatRaster-method (layerCor),
144

- legend, 22, 179
- length, 13, 16
- length (dimensions), 80
- length, SpatRasterCollection-method
(dimensions), 80
- length, SpatRasterDataset-method
(dimensions), 80
- length, SpatVector-method (dimensions),
80
- length, SpatVectorCollection-method
(dimensions), 80
- levels, 12, 20, 139, 140
- levels (factors), 99
- levels, SpatRaster-method (factors), 99
- levels<- (factors), 99
- levels<- , SpatRaster-method (factors), 99
- linearUnits, 14, 145
- linearUnits, SpatRaster-method
(linearUnits), 145
- linearUnits, SpatVector-method
(linearUnits), 145
- lines, 17, 146, 178, 180, 181, 184, 186
- lines, leaflet-method (plet), 176
- lines, sf-method (lines), 146
- lines, SpatExtent-method (lines), 146
- lines, SpatGraticule, missing-method
(plot_graticule), 185
- lines, SpatGraticule-method (lines), 146
- lines, SpatRaster-method (lines), 146
- lines, SpatVector-method (lines), 146
- log (Math-methods), 154
- log, SpatRaster-method (Math-methods),
154
- logic (Compare-methods), 61
- Logic, logical, SpatRaster-method
(Compare-methods), 61
- Logic, numeric, SpatRaster-method
(Compare-methods), 61
- Logic, SpatRaster, logical-method
(Compare-methods), 61
- Logic, SpatRaster, numeric-method
(Compare-methods), 61
- Logic, SpatRaster, SpatRaster-method
(Compare-methods), 61
- logic, SpatRaster-method
(Compare-methods), 61
- Logic-methods, 8
- Logic-methods (Compare-methods), 61

- longnames (names), 163
- longnames, SpatRaster-method (names), 163
- longnames, SpatRasterDataset-method (names), 163
- longnames<- (names), 163
- longnames<-, SpatRaster-method (names), 163
- longnames<-, SpatRasterDataset-method (names), 163
- make.unique, 164
- makeNodes, 15
- makeNodes (topology), 267
- makeNodes, SpatVector-method (topology), 267
- makeTiles, 148, 282
- makeTiles, SpatRaster-method (makeTiles), 148
- makeValid, 16, 237
- makeValid (valid), 274
- makeValid, SpatVector-method (valid), 274
- makeVRT, 149, 282
- map.pal, 150
- mask, 9, 69, 90, 124, 151, 201
- mask, SpatRaster, sf-method (mask), 151
- mask, SpatRaster, SpatRaster-method (mask), 151
- mask, SpatRaster, SpatVector-method (mask), 151
- mask, SpatVector, sf-method (mask), 151
- mask, SpatVector, SpatVector-method (mask), 151
- match, 153, 153
- match, SpatRaster-method (match), 153
- math, 142
- math (Math-methods), 154
- Math, SpatExtent-method (Math-methods), 154
- Math, SpatRaster-method (Math-methods), 154
- math, SpatRaster-method (Math-methods), 154
- Math-methods, 8, 16, 154
- Math2, SpatExtent-method (Math-methods), 154
- Math2, SpatRaster-method (Math-methods), 154
- Math2, SpatVector-method (Math-methods), 154
- Math2-methods (Math-methods), 154
- max (summarize), 254
- max, SpatRaster-method (summarize), 254
- mean (summarize), 254
- mean, SpatExtent-method (summarize), 254
- mean, SpatRaster-method (summarize), 254
- mean, SpatVector-method (summarize), 254
- median (summarize), 254
- median, SpatRaster-method (summarize), 254
- median, SpatVector-method (summarize), 254
- mem, 155
- mem_info, 13, 169, 291
- mem_info (mem), 155
- merge, 8, 13, 15, 95, 156, 156, 160, 246, 270
- merge, SpatRaster, SpatRaster-method (merge), 156
- merge, SpatRasterCollection, missing-method (merge), 156
- merge, SpatVector, data.frame-method (merge), 156
- merge, SpatVector, SpatVector-method (merge), 156
- mergeLines, 15
- mergeLines (topology), 267
- mergeLines, SpatVector-method (topology), 267
- mergeTime, 11, 157
- mergeTime, SpatRasterDataset-method (mergeTime), 157
- meta, 158
- meta, SpatRaster-method (meta), 158
- min (summarize), 254
- min, SpatRaster-method (summarize), 254
- minmax, 10
- minmax (extremes), 98
- minmax, SpatRaster-method (extremes), 98
- minRect, 286
- minRect (convHull), 65
- minRect, SpatVector-method (convHull), 65
- modal, 159, 254, 256
- modal, SpatRaster-method (modal), 159
- mosaic, 8, 13, 157, 160, 246, 270
- mosaic, SpatRaster, SpatRaster-method (mosaic), 160
- mosaic, SpatRasterCollection, missing-method (mosaic), 160

- na.omit, [13](#), [161](#)
- na.omit, SpatVector-method (na.omit), [161](#)
- NAflag, [11](#), [19](#), [162](#)
- NAflag, SpatRaster-method (NAflag), [162](#)
- NAflag<- (NAflag), [162](#)
- NAflag<-, SpatRaster-method (NAflag), [162](#)
- name (names), [163](#)
- name<- (names), [163](#)
- names, [11](#), [13](#), [14](#), [123](#), [163](#), [186](#), [273](#)
- names, SpatRaster-method (names), [163](#)
- names, SpatRasterCollection-method (names), [163](#)
- names, SpatRasterDataset-method (names), [163](#)
- names, SpatVector-method (names), [163](#)
- names, SpatVectorCollection-method (names), [163](#)
- names, SpatVectorProxy-method (names), [163](#)
- names<- (names), [163](#)
- names<-, SpatRaster-method (names), [163](#)
- names<-, SpatRasterCollection-method (names), [163](#)
- names<-, SpatRasterDataset-method (names), [163](#)
- names<-, SpatVector-method (names), [163](#)
- names<-, SpatVectorCollection-method (names), [163](#)
- ncell, [11](#), [296](#)
- ncell (dimensions), [80](#)
- ncell, ANY-method (dimensions), [80](#)
- ncell, SpatRaster-method (dimensions), [80](#)
- ncell, SpatRasterDataset-method (dimensions), [80](#)
- ncol, [11](#), [14](#)
- ncol (dimensions), [80](#)
- ncol, SpatRaster-method (dimensions), [80](#)
- ncol, SpatRasterCollection-method (dimensions), [80](#)
- ncol, SpatRasterDataset-method (dimensions), [80](#)
- ncol, SpatVector-method (dimensions), [80](#)
- ncol<- (dimensions), [80](#)
- ncol<-, SpatRaster, numeric-method (dimensions), [80](#)
- ncvar_def, [289](#)
- nearby, [14](#), [23](#), [208](#)
- nearby (nearest), [165](#)
- nearby, SpatVector-method (nearest), [165](#)
- nearest, [14](#), [165](#)
- nearest, SpatVector-method (nearest), [165](#)
- nlyr, [11](#), [19](#)
- nlyr (dimensions), [80](#)
- nlyr, SpatRaster-method (dimensions), [80](#)
- nlyr, SpatRasterCollection-method (dimensions), [80](#)
- nlyr, SpatRasterDataset-method (dimensions), [80](#)
- nlyr<- (dimensions), [80](#)
- nlyr<-, SpatRaster, numeric-method (dimensions), [80](#)
- noNA, [61](#)
- noNA (summarize), [254](#)
- noNA, SpatRaster-method (summarize), [254](#)
- normalize.longitude, [16](#), [166](#), [219](#)
- normalize.longitude, SpatVector-method (normalize.longitude), [166](#)
- north, [17](#), [166](#), [181](#), [186](#), [222](#)
- not.na, [8](#), [61](#), [168](#)
- not.na, SpatRaster-method (not.na), [168](#)
- nrow, [11](#), [14](#)
- nrow (dimensions), [80](#)
- nrow, SpatRaster-method (dimensions), [80](#)
- nrow, SpatRasterCollection-method (dimensions), [80](#)
- nrow, SpatRasterDataset-method (dimensions), [80](#)
- nrow, SpatVector-method (dimensions), [80](#)
- nrow<- (dimensions), [80](#)
- nrow<-, SpatRaster, numeric-method (dimensions), [80](#)
- nsrc (dimensions), [80](#)
- nsrc, SpatRaster-method (dimensions), [80](#)
- options, [169](#)
- origin, [11](#), [170](#)
- origin, SpatRaster-method (origin), [170](#)
- origin<- (origin), [170](#)
- origin<-, SpatRaster-method (origin), [170](#)
- PackedSpatRaster-class (SpatRaster-class), [241](#)
- PackedSpatVector-class (SpatVector-class), [243](#)
- pairs, [18](#), [43](#), [123](#), [171](#), [171](#)
- pairs, SpatRaster-method (pairs), [171](#)
- panel, [17](#), [172](#), [181](#)

- panel, SpatRaster-method (panel), 172
- par, 147, 177
- patches, 9, 18, 42, 173
- patches, SpatRaster-method (patches), 173
- perim, 14, 174
- perim, SpatVector-method (perim), 174
- perimeter (perim), 174
- perimeter, SpatVector-method (perim), 174
- persp, 17, 175, 175
- persp, SpatRaster-method (persp), 175
- plet, 176
- plet, missing-method (plet), 176
- plet, SpatRaster-method (plet), 176
- plet, SpatVector-method (plet), 176
- plet, SpatVectorCollection-method (plet), 176
- plot, 17, 18, 28, 47, 65, 76, 87, 122, 125, 167, 172, 177, 178, 181, 183–186, 216, 222, 264, 299, 300
- plot, SpatExtent, missing-method (plot_extent), 184
- plot, SpatGraticule, missing-method (plot_graticule), 185
- plot, SpatRaster, character-method (plot), 178
- plot, SpatRaster, missing-method (plot), 178
- plot, SpatRaster, numeric-method (plot), 178
- plot, SpatRaster, SpatRaster-method (scatterplot), 224
- plot, SpatVector, character-method (plot), 178
- plot, SpatVector, data.frame-method (plot), 178
- plot, SpatVector, missing-method (plot), 178
- plot, SpatVector, numeric-method (plot), 178
- plot, SpatVectorProxy, missing-method (plot), 178
- plot<SpatGraticule>, 16, 17, 120
- plot_extent, 184
- plot_graticule, 185
- plotRGB, 17, 181, 182, 216
- plotRGB, SpatRaster-method (plotRGB), 182
- points, 17, 87, 147, 178, 181, 186
- points (lines), 146
- points, leaflet-method (plet), 176
- points, sf-method (lines), 146
- points, SpatExtent-method (lines), 146
- points, SpatRaster-method (lines), 146
- points, SpatVector-method (lines), 146
- polys, 17, 178, 181, 186
- polys (lines), 146
- polys, sf-method (lines), 146
- polys, SpatExtent-method (lines), 146
- polys, SpatRaster-method (lines), 146
- polys, SpatVector-method (lines), 146
- POSIXlt, 265
- predict, 10, 55, 134, 135, 186
- predict, SpatRaster-method (predict), 186
- prod (summarize), 254
- prod, SpatRaster-method (summarize), 254
- project, 8, 10, 13, 19, 72, 174, 189, 214, 215
- project, matrix-method (project), 189
- project, SpatExtent-method (project), 189
- project, SpatRaster-method (project), 189
- project, SpatVector-method (project), 189
- quantile, 9, 19, 192, 257, 258
- quantile, SpatRaster-method (quantile), 192
- quantile, SpatVector-method (quantile), 192
- query, 193
- query, SpatVectorProxy-method (query), 193
- rainbow, 41
- range (summarize), 254
- range, SpatRaster-method (summarize), 254
- rangeFill, 9, 194
- rangeFill, SpatRaster-method (rangeFill), 194
- rapp, 8, 195, 195, 230
- rapp, SpatRaster-method (rapp), 195
- rast, 7, 17, 18, 197, 241
- rast, ANY-method (rast), 197
- rast, array-method (rast), 197
- rast, character-method (rast), 197
- rast, data.frame-method (rast), 197
- rast, list-method (rast), 197
- rast, matrix-method (rast), 197
- rast, missing-method (rast), 197
- rast, PackedSpatRaster-method (rast), 197
- rast, SpatExtent-method (rast), 197

- `rast`, `SpatRaster`-method (`rast`), 197
- `rast`, `SpatRasterDataset`-method (`rast`), 197
- `rast`, `SpatVector`-method (`rast`), 197
- `rast`, `stars`-method (`rast`), 197
- `rast`, `stars_proxy`-method (`rast`), 197
- `rasterImage`, 36, 184
- `rasterize`, 17, 96, 132, 134, 200, 202–204
- `rasterize`, `matrix`, `SpatRaster`-method (`rasterize`), 200
- `rasterize`, `sf`, `SpatRaster`-method (`rasterize`), 200
- `rasterize`, `SpatVector`, `SpatRaster`-method (`rasterize`), 200
- `rasterizeGeom`, 17, 201, 201, 204
- `rasterizeGeom`, `SpatVector`, `SpatRaster`-method (`rasterizeGeom`), 201
- `rasterizeWin`, 17, 132, 134, 201, 203
- `rasterizeWin`, `data.frame`, `SpatRaster`-method (`rasterizeWin`), 203
- `rasterizeWin`, `SpatVector`, `SpatRaster`-method (`rasterizeWin`), 203
- `RasterSource` (`SpatRaster`-class), 241
- `RasterSource`-class (`SpatRaster`-class), 241
- `rcl`, 204
- `rcl`, `SpatRaster`-method (`rcl`), 204
- `Rcpp_RasterSource`-class (`SpatRaster`-class), 241
- `Rcpp_SpatCategories`-class (`SpatRaster`-class), 241
- `Rcpp_SpatExtent`-class (`SpatExtent`-class), 240
- `Rcpp_SpatRaster`-class (`SpatRaster`-class), 241
- `Rcpp_SpatVector`-class (`SpatVector`-class), 243
- `readRDS` (`serialize`), 231
- `readRDS`, `character`-method (`serialize`), 231
- `readStart`, 12
- `readStart` (`readwrite`), 205
- `readStart`, `SpatRaster`-method (`readwrite`), 205
- `readStart`, `SpatRasterDataset`-method (`readwrite`), 205
- `readStop`, 12
- `readStop` (`readwrite`), 205
- `readStop`, `SpatRaster`-method (`readwrite`), 205
- `readStop`, `SpatRasterDataset`-method (`readwrite`), 205
- `readValues` (`readwrite`), 205
- `readValues`, `SpatRaster`-method (`readwrite`), 205
- `readwrite`, 205
- `rectify`, 207
- `rectify`, `SpatRaster`-method (`rectify`), 207
- `relate`, 14, 23, 138, 165, 208
- `relate`, `SpatExtent`, `SpatExtent`-method (`relate`), 208
- `relate`, `SpatExtent`, `SpatRaster`-method (`relate`), 208
- `relate`, `SpatExtent`, `SpatVector`-method (`relate`), 208
- `relate`, `SpatRaster`, `SpatExtent`-method (`relate`), 208
- `relate`, `SpatRaster`, `SpatRaster`-method (`relate`), 208
- `relate`, `SpatRaster`, `SpatVector`-method (`relate`), 208
- `relate`, `SpatVector`, `missing`-method (`relate`), 208
- `relate`, `SpatVector`, `SpatExtent`-method (`relate`), 208
- `relate`, `SpatVector`, `SpatRaster`-method (`relate`), 208
- `relate`, `SpatVector`, `SpatVector`-method (`relate`), 208
- `removeDupNodes`, 15
- `removeDupNodes` (`topology`), 267
- `removeDupNodes`, `SpatVector`-method (`topology`), 267
- `rep`, 210, 210
- `rep`, `SpatRaster`-method (`rep`), 210
- `replace_dollar`, 211
- `replace_layers`, 212
- `replace_values`, 213
- `res`, 11, 198
- `res` (`dimensions`), 80
- `res`, `SpatRaster`-method (`dimensions`), 80
- `res`, `SpatRasterDataset`-method (`dimensions`), 80
- `res<-` (`dimensions`), 80
- `res<-`, `SpatRaster`, `numeric`-method (`dimensions`), 80

- res<-, SpatRaster-method (dimensions), 80
- resample, 8, 26, 69, 83, 126, 191, 207, 214
- resample, SpatRaster, SpatRaster-method (resample), 214
- rescale, 15, 47, 93, 131, 215, 244
- rescale, SpatRaster-method (rescale), 215
- rescale, SpatVector-method (rescale), 215
- rev (flip), 103
- rev, SpatRaster-method (flip), 103
- RGB, 184, 216
- RGB, SpatRaster-method (RGB), 216
- RGB<- (RGB), 216
- RGB<-, SpatRaster-method (RGB), 216
- roll, 9, 19, 30, 155, 217
- roll, numeric-method (roll), 217
- roll, SpatRaster-method (roll), 217
- rotate, 8, 16, 103, 166, 216, 219, 236, 269
- rotate, SpatRaster-method (rotate), 219
- rotate, SpatVector-method (rotate), 219
- round, 41
- round (Math-methods), 154
- round, SpatRaster-method (Math-methods), 154
- round, SpatVector-method (Math-methods), 154
- rowColCombine, 205
- rowColCombine (xyRowColCell), 294
- rowColCombine, SpatRaster, numeric, numeric-method (xyRowColCell), 294
- rowColFromCell, 11
- rowColFromCell (xyRowColCell), 294
- rowColFromCell, SpatRaster, numeric-method (xyRowColCell), 294
- rowFromCell (xyRowColCell), 294
- rowFromCell, SpatRaster, numeric-method (xyRowColCell), 294
- rowFromY, 11
- rowFromY (xyRowColCell), 294
- rowFromY, SpatRaster, numeric-method (xyRowColCell), 294
- runif, 127
- same.crs, 220
- sapp, 8, 142, 221
- sapp, SpatRaster-method (sapp), 221
- sapp, SpatRasterDataset-method (sapp), 221
- saveRDS, 231, 232
- saveRDS (serialize), 231
- saveRDS, SpatRaster-method (serialize), 231
- saveRDS, SpatVector-method (serialize), 231
- sbar, 17, 131, 167, 181, 186, 222
- scale, 9, 223, 223, 224
- scale, SpatRaster-method (scale), 223
- scatterplot, 224
- scoff, 225, 291
- scoff, SpatRaster-method (scoff), 225
- scoff<- (scoff), 225
- scoff<-, SpatRaster-method (scoff), 225
- sds, 13, 199, 226, 246
- sds, array-method (sds), 226
- sds, character-method (sds), 226
- sds, list-method (sds), 226
- sds, missing-method (sds), 226
- sds, SpatRaster-method (sds), 226
- sds, stars-method (sds), 226
- sds, stars_proxy-method (sds), 226
- segregate, 9, 19, 227
- segregate, SpatRaster-method (segregate), 227
- sel, 15, 18, 228
- sel, SpatRaster-method (sel), 228
- sel, SpatVector-method (sel), 228
- selectHighest, 229
- selectHighest, SpatRaster-method (selectHighest), 229
- selectRange, 7, 19, 195, 196, 230
- selectRange, SpatRaster-method (selectRange), 230
- serialize, 231, 231, 232
- serialize, SpatRaster-method (serialize), 231
- serialize, SpatVector-method (serialize), 231
- set.cats, 12, 100
- set.cats (inplace), 128
- set.cats, SpatRaster-method (inplace), 128
- set.crs (inplace), 128
- set.crs, SpatRaster-method (inplace), 128
- set.crs, SpatVector-method (inplace), 128
- set.ext, 93
- set.ext (inplace), 128
- set.ext, SpatRaster-method (inplace), 128
- set.ext, SpatVector-method (inplace), 128

- set.names, [163](#)
- set.names (inplace), [128](#)
- set.names, SpatRaster-method (inplace), [128](#)
- set.names, SpatRasterCollection-method (inplace), [128](#)
- set.names, SpatRasterDataset-method (inplace), [128](#)
- set.names, SpatVector-method (inplace), [128](#)
- set.names, SpatVectorCollection-method (inplace), [128](#)
- set.RGB (inplace), [128](#)
- set.RGB, SpatRaster-method (inplace), [128](#)
- set.values, [213](#), [231](#)
- set.values (inplace), [128](#)
- set.values, SpatRaster-method (inplace), [128](#)
- setCats (deprecated), [77](#)
- setCats, SpatRaster-method (deprecated), [77](#)
- setGDALconfig, [282](#)
- setGDALconfig (gdal), [115](#)
- setMinMax, [10](#)
- setMinMax (extremes), [98](#)
- setMinMax, SpatRaster-method (extremes), [98](#)
- setValues, [10](#), [232](#)
- setValues, SpatRaster, ANY-method (setValues), [232](#)
- setValues, SpatRaster-method (setValues), [232](#)
- setValues, SpatVector, ANY-method (setValues), [232](#)
- setValues, SpatVector-method (setValues), [232](#)
- shade, [10](#), [233](#)
- sharedPaths, [15](#), [60](#), [115](#), [234](#), [237](#), [268](#)
- sharedPaths, SpatVector-method (sharedPaths), [234](#)
- shift, [8](#), [15](#), [131](#), [216](#), [219](#), [235](#), [244](#)
- shift, SpatExtent-method (shift), [235](#)
- shift, SpatRaster-method (shift), [235](#)
- shift, SpatVector-method (shift), [235](#)
- show, [123](#)
- show, SpatExtent-method (SpatExtent-class), [240](#)
- show, SpatRaster-method (SpatRaster-class), [241](#)
- show, SpatVector-method (SpatVector-class), [243](#)
- sieve, [10](#), [236](#)
- sieve, SpatRaster-method (sieve), [236](#)
- simplifyGeom, [15](#), [237](#), [268](#)
- simplifyGeom, SpatVector-method (simplifyGeom), [237](#)
- size (dimensions), [80](#)
- size, SpatRaster-method (dimensions), [80](#)
- snap, [16](#)
- snap (topology), [267](#)
- snap, SpatVector-method (topology), [267](#)
- sort, [15](#), [238](#)
- sort, data.frame-method (sort), [238](#)
- sort, SpatRaster-method (sort), [238](#)
- sort, SpatVector-method (sort), [238](#)
- sources, [11](#), [12](#), [239](#)
- sources, SpatRaster-method (sources), [239](#)
- sources, SpatRasterCollection-method (sources), [239](#)
- sources, SpatRasterDataset-method (sources), [239](#)
- sources, SpatVector-method (sources), [239](#)
- sources, SpatVectorProxy-method (sources), [239](#)
- SpatCategories (SpatRaster-class), [241](#)
- SpatCategories-class (SpatRaster-class), [241](#)
- SpatExtent, [94](#), [178](#), [181](#), [183](#), [226](#)
- SpatExtent (SpatExtent-class), [240](#)
- SpatExtent-class, [240](#)
- SpatGraticule, [178](#), [181](#)
- SpatRaster (SpatRaster-class), [241](#)
- SpatRaster-class, [241](#)
- SpatRasterCollection (SpatRaster-class), [241](#)
- SpatRasterCollection-class (SpatRaster-class), [241](#)
- SpatRasterDataset (SpatRaster-class), [241](#)
- SpatRasterDataset-class (SpatRaster-class), [241](#)
- spatSample, [10](#), [19](#), [241](#), [258](#)
- spatSample, SpatExtent-method (spatSample), [241](#)
- spatSample, SpatRaster-method (spatSample), [241](#)

- spatSample, SpatVector-method
(spatSample), 241
- SpatVector (SpatVector-class), 243
- SpatVector-class, 243
- SpatVectorCollection
(SpatVector-class), 243
- SpatVectorCollection-class
(SpatVector-class), 243
- SpatVectorProxy (SpatVector-class), 243
- SpatVectorProxy-class
(SpatVector-class), 243
- spin, 15, 219, 244
- spin, SpatVector-method (spin), 244
- split, 245
- split, SpatRaster, ANY-method (split), 245
- split, SpatVector, ANY-method (split), 245
- split, SpatVector, SpatVector-method
(split), 245
- sprc, 13, 246
- sprc, character-method (sprc), 246
- sprc, list-method (sprc), 246
- sprc, missing-method (sprc), 246
- sprc, SpatRaster-method (sprc), 246
- sqrt (Math-methods), 154
- sqrt, SpatRaster-method (Math-methods),
154
- stdev (summarize), 254
- stdev, SpatRaster-method (summarize), 254
- stretch, 9, 247
- stretch, SpatRaster-method (stretch), 247
- subset, 7, 19, 195, 248
- subset, SpatRaster-method (subset), 248
- subset, SpatVector-method (subset), 248
- subset_dollar, 249
- subset_double, 251
- subset_single, 252
- subst, 9, 55, 213, 253
- subst, SpatRaster-method (subst), 253
- sum (summarize), 254
- sum, SpatRaster-method (summarize), 254
- summarize, 254
- summary, 9, 257, 257, 258
- Summary, SpatExtent-method (summary), 257
- Summary, SpatRaster-method (summary), 257
- summary, SpatRaster-method (summary), 257
- Summary, SpatVector-method (summary), 257
- summary, SpatVector-method (summary), 257
- Summary-methods, 8, 19
- Summary-methods (summarize), 254
- svc, 16, 258
- svc, list-method (svc), 258
- svc, missing-method (svc), 258
- svc, sf-method (svc), 258
- svc, SpatVector-method (svc), 258
- syndif, 14, 259
- syndif, SpatVector, SpatVector-method
(syndif), 259
- t, 8, 15, 216, 244
- t (transpose), 268
- t, SpatRaster-method (transpose), 268
- t, SpatVector-method (transpose), 268
- tail (headtail), 122
- tail, SpatRaster-method (headtail), 122
- tail, SpatVector-method (headtail), 122
- tapp, 8, 19, 30, 142, 196, 221, 230, 260
- tapp, SpatRaster-method (tapp), 260
- tapply, 260
- terra (terra-package), 7
- terra-package, 7
- terrain, 10, 221, 233, 234, 261, 280
- terrain, SpatRaster-method (terrain), 261
- terrain.colors, 151
- terraOptions, 12, 267, 291
- terraOptions (options), 169
- text, 17, 122, 228, 263, 264
- text, SpatRaster-method (text), 263
- text, SpatVector-method (text), 263
- tighten, 264
- tighten, SpatRaster-method (tighten), 264
- tighten, SpatRasterDataset-method
(tighten), 264
- tiles, 282
- tiles (makeTiles), 148
- tiles, SpatRaster-method (makeTiles), 148
- time, 11, 78, 260, 265, 273
- time, SpatRaster-method (time), 265
- time<- (time), 265
- time<-, SpatRaster-method (time), 265
- timeInfo (time), 265
- timeInfo, SpatRaster-method (time), 265
- tmpFiles, 12, 266
- topology, 115, 235, 267
- trans, 103
- trans (transpose), 268
- trans, SpatRaster-method (transpose), 268
- transpose, 268

- Trig, [38](#)
- trim, [8](#), [269](#)
- trim, SpatRaster-method (trim), [269](#)
- union, [14](#), [16](#), [60](#), [138](#), [157](#), [270](#)
- union, SpatExtent, SpatExtent-method (union), [270](#)
- union, SpatVector, missing-method (union), [270](#)
- union, SpatVector, SpatExtent-method (union), [270](#)
- union, SpatVector, SpatVector-method (union), [270](#)
- unique, [9](#), [13](#), [271](#), [271](#)
- unique, SpatRaster, ANY-method (unique), [271](#)
- unique, SpatRaster-method (unique), [271](#)
- unique, SpatVector, ANY-method (unique), [271](#)
- unique, SpatVector-method (unique), [271](#)
- units, [272](#)
- units, SpatRaster-method (units), [272](#)
- units, SpatRasterDataset-method (units), [272](#)
- units<- (units), [272](#)
- units<- , SpatRaster-method (units), [272](#)
- units<- , SpatRasterDataset-method (units), [272](#)
- unserialize (serialize), [231](#)
- unserialize, ANY-method (serialize), [231](#)
- unwrap (wrap), [288](#)
- unwrap, ANY-method (wrap), [288](#)
- unwrap, PackedSpatRaster-method (wrap), [288](#)
- unwrap, PackedSpatVector-method (wrap), [288](#)
- update, [273](#)
- update, SpatRaster-method (update), [273](#)
- valid, [274](#)
- values, [10](#), [15](#), [19](#), [97](#), [213](#), [233](#), [275](#)
- values, SpatRaster-method (values), [275](#)
- values, SpatVector-method (values), [275](#)
- values<- , [10](#), [15](#)
- values<- (setValues), [232](#)
- values<- , SpatRaster, ANY-method (setValues), [232](#)
- values<- , SpatVector, ANY-method (setValues), [232](#)
- values<- , SpatVector, data.frame-method (setValues), [232](#)
- values<- , SpatVector, matrix-method (setValues), [232](#)
- values<- , SpatVector, NULL-method (setValues), [232](#)
- varnames (names), [163](#)
- varnames, SpatRaster-method (names), [163](#)
- varnames, SpatRasterDataset-method (names), [163](#)
- varnames<- (names), [163](#)
- varnames<- , SpatRaster-method (names), [163](#)
- varnames<- , SpatRasterDataset-method (names), [163](#)
- vect, [13](#), [17](#), [19](#), [194](#), [199](#), [276](#)
- vect, character-method (vect), [276](#)
- vect, data.frame-method (vect), [276](#)
- vect, list-method (vect), [276](#)
- vect, matrix-method (vect), [276](#)
- vect, missing-method (vect), [276](#)
- vect, PackedSpatVector-method (vect), [276](#)
- vect, sf-method (vect), [276](#)
- vect, sfc-method (vect), [276](#)
- vect, SpatExtent-method (vect), [276](#)
- vect, Spatial-method (vect), [276](#)
- vect, SpatVector-method (vect), [276](#)
- vect, SpatVectorCollection-method (vect), [276](#)
- vect, XY-method (vect), [276](#)
- vector_layers, [13](#), [279](#), [292](#)
- viewshed, [10](#), [263](#), [280](#)
- viewshed, SpatRaster-method (viewshed), [280](#)
- voronoi, [13](#), [281](#)
- voronoi, SpatVector-method (voronoi), [281](#)
- vrt, [148](#), [150](#), [157](#), [282](#)
- vrt, character-method (vrt), [282](#)
- weighted.mean, [145](#), [283](#), [284](#)
- weighted.mean, SpatRaster, numeric-method (weighted.mean), [283](#)
- weighted.mean, SpatRaster, SpatRaster-method (weighted.mean), [283](#)
- where, [284](#)
- which, [284](#), [285](#)
- which.lyr, [9](#), [256](#), [285](#)
- which.lyr, SpatRaster-method (which.lyr), [285](#)

- `which.max (summarize)`, 254
- `which.max, SpatRaster-method (summarize)`, 254
- `which.min (summarize)`, 254
- `which.min, SpatRaster-method (summarize)`, 254
- `width`, 15, 286
- `width, SpatVector-method (width)`, 286
- `window`, 198, 287
- `window, SpatRaster-method (window)`, 287
- `window<- (window)`, 287
- `window<-, SpatRaster-method (window)`, 287
- `wrap`, 7, 199, 231, 288
- `wrap, SpatRaster-method (wrap)`, 288
- `wrap, SpatVector-method (wrap)`, 288
- `writeCDF`, 12, 289, 291
- `writeCDF, SpatRaster-method (writeCDF)`, 289
- `writeCDF, SpatRasterDataset-method (writeCDF)`, 289
- `writeRaster`, 7, 12, 24, 29, 32, 39, 42, 44, 47, 50, 52, 53, 55, 61, 63, 66, 67, 70, 79, 82, 83, 85, 95, 102, 103, 105, 107, 108, 111, 112, 121, 124, 126, 127, 132, 133, 135, 140, 142, 148, 152, 154, 156, 158–160, 168, 169, 173, 187, 191, 192, 195, 196, 199, 201, 202, 204, 207, 214, 217–219, 221, 227, 230, 231, 234, 236, 238, 247, 248, 254, 256, 261, 262, 269, 280, 283, 290, 290, 297
- `writeRaster, SpatRaster, character-method (writeRaster)`, 290
- `writeStart`, 12
- `writeStart (readwrite)`, 205
- `writeStart, SpatRaster, character-method (readwrite)`, 205
- `writeStop`, 12
- `writeStop (readwrite)`, 205
- `writeStop, SpatRaster-method (readwrite)`, 205
- `writeValues`, 12
- `writeValues (readwrite)`, 205
- `writeValues, SpatRaster, vector-method (readwrite)`, 205
- `writeVector`, 13, 292
- `writeVector, SpatVector, character-method (writeVector)`, 292
- `xFromCell`, 11
- `xFromCell (xyRowColCell)`, 294
- `xFromCell, SpatRaster, numeric-method (xyRowColCell)`, 294
- `xFromCol`, 11
- `xFromCol (xyRowColCell)`, 294
- `xFromCol, SpatRaster, missing-method (xyRowColCell)`, 294
- `xFromCol, SpatRaster, numeric-method (xyRowColCell)`, 294
- `xmax`, 11
- `xmax (xmin)`, 293
- `xmax, SpatExtent-method (xmin)`, 293
- `xmax, SpatRaster-method (xmin)`, 293
- `xmax, SpatVector-method (xmin)`, 293
- `xmax<- (xmin)`, 293
- `xmax<-, SpatExtent, numeric-method (xmin)`, 293
- `xmax<-, SpatRaster, numeric-method (xmin)`, 293
- `xmin`, 11, 293
- `xmin, SpatExtent-method (xmin)`, 293
- `xmin, SpatRaster-method (xmin)`, 293
- `xmin, SpatVector-method (xmin)`, 293
- `xmin<- (xmin)`, 293
- `xmin<-, SpatExtent, numeric-method (xmin)`, 293
- `xmin<-, SpatRaster, numeric-method (xmin)`, 293
- `xres`, 11
- `xres (dimensions)`, 80
- `xres, SpatRaster-method (dimensions)`, 80
- `xyFromCell`, 11, 69, 96, 117
- `xyFromCell (xyRowColCell)`, 294
- `xyFromCell, SpatRaster, numeric-method (xyRowColCell)`, 294
- `xyRowColCell`, 294
- `yFromCell`, 11
- `yFromCell (xyRowColCell)`, 294
- `yFromCell, SpatRaster, numeric-method (xyRowColCell)`, 294
- `yFromRow`, 11
- `yFromRow (xyRowColCell)`, 294
- `yFromRow, SpatRaster, missing-method (xyRowColCell)`, 294
- `yFromRow, SpatRaster, numeric-method (xyRowColCell)`, 294
- `ymax`, 11

`ymin(xmin)`, [293](#)
`ymin, SpatExtent-method(xmin)`, [293](#)
`ymin, SpatRaster-method(xmin)`, [293](#)
`ymin, SpatVector-method(xmin)`, [293](#)
`ymin<-(xmin)`, [293](#)
`ymin<-, SpatExtent,numeric-method(xmin)`, [293](#)
`ymin<-, SpatRaster,numeric-method(xmin)`, [293](#)
`ymin`, [11](#)
`ymin(xmin)`, [293](#)
`ymin, SpatExtent-method(xmin)`, [293](#)
`ymin, SpatRaster-method(xmin)`, [293](#)
`ymin, SpatVector-method(xmin)`, [293](#)
`ymin<-(xmin)`, [293](#)
`ymin<-, SpatExtent,numeric-method(xmin)`, [293](#)
`ymin<-, SpatRaster,numeric-method(xmin)`, [293](#)
`yres`, [11](#)
`yres(dimensions)`, [80](#)
`yres, SpatRaster-method(dimensions)`, [80](#)

`zonal`, [9](#), [71](#), [92](#), [96](#), [97](#), [119](#), [297](#)
`zonal, SpatRaster, SpatRaster-method(zonal)`, [297](#)
`zonal, SpatRaster, SpatVector-method(zonal)`, [297](#)
`zonal, SpatVector, SpatVector-method(zonal)`, [297](#)
`zoom`, [18](#), [299](#)
`zoom, SpatRaster-method(zoom)`, [299](#)
`zoom, SpatVector-method(zoom)`, [299](#)