Package: git2rdata (via r-universe)

July 12, 2024

Title Store and Retrieve Data.frames in a Git Repository

Version 0.4.0

Description The git2rdata package is an R package for writing and reading dataframes as plain text files. A metadata file stores important information. 1) Storing metadata allows to maintain the classes of variables. By default, git2rdata optimizes the data for file storage. The optimization is most effective on data containing factors. The optimization makes the data less human readable. The user can turn this off when they prefer a human readable format over smaller files. Details on the implementation are available in vignette(``plain_text", package = ``git2rdata"). 2) Storing metadata also allows smaller row based diffs between two consecutive commits. This is a useful feature when storing data as plain text files under version control. Details on this part of the implementation are available in vignette(``version_control", package = ``git2rdata"). Although we envisioned git2rdata with a git workflow in mind, you can use it in combination with other version control systems like subversion or mercurial. 3) git2rdata is a useful tool in a reproducible and traceable workflow. vignette(``workflow", package = ``git2rdata") gives a toy example. 4) vignette(``efficiency", package = ``git2rdata") provides some insight into the efficiency of file storage, git repository size and speed for writing and reading.

License GPL-3

URL https://ropensci.github.io/git2rdata/, https://github.com/ropensci/git2rdata/

BugReports https://github.com/ropensci/git2rdata/issues

Depends R (>= 3.5.0)

Imports assertthat, git2r (>= 0.23.0), methods, yaml

Suggests ggplot2, knitr, microbenchmark, rmarkdown, spelling, testthat

VignetteBuilder knitr

Encoding UTF-8

Language eng

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Collate 'clean_data_path.R' 'datahash.R' 'git2rdata_package.R' 'write_vc.R' 'is_git2rdata.R' 'is_git2rmeta.R' 'list_data.R' 'meta.R' 'prune.R' 'read_vc.R' 'recent_commit.R' 'reexport.R' 'relabel.R' 'rename_variable.R' 'upgrade_data.R' 'utils.R' 'verify_vc.R'

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/git2rdata

RemoteRef main

RemoteSha f11496864dbe6bf3538febfd2f23c8940d47af0b

Contents

`commit` *Re-exported Function From* `git2r`

Description

See `commit` in `git2r`.

See Also

Other version_control: `pull()`, `push()`, `recent_commit()`, `repository()`, `status()`

`list_data` *List Available Git2rdata Files Containing Data*

Description

The function returns the names of all valid git2rdata objects. This implies `.tsv` files with a matching **valid** metadata file (`.yml`). **Invalid** metadata files result in a warning. The function ignores **valid** metadata files without matching raw data (`.tsv`).

Usage

```
list_data(root = ".", path = ".", recursive = TRUE)
```

Arguments

`root`	the `root` of the repository. Either a path or a `git-repository`
`path`	relative `path` from the `root`. Defaults to the `root`
`recursive`	logical. Should the listing recurse into directories?

Value

A character vector of git2rdata object names, including their relative path.

See Also

Other storage: `prune_meta()`, `read_vc()`, `relabel()`, `rename_variable()`, `rm_data()`, `verify_vc()`, `write_vc()`

Examples

```
## on file system

# create a directory
root <- tempfile("git2rdata-")
dir.create(root)

# store a dataframe as git2rdata object. Capture the result to minimise
# screen output
junk <- write_vc(iris[1:6, ], "iris", root, sorting = "Sepal.Length")
# write a standard tab separate file (non git2rdata object)
write.table(iris, file = file.path(root, "standard.tsv"), sep = "\t")
# write a YAML file
yml <- list(
  authors = list(
   "Research Institute for Nature and Forest" = list(
       href = "https://www.inbo.be/en")))
yaml::write_yaml(yml, file = file.path(root, "_pkgdown.yml"))
```

```
# list the git2rdata objects
list_data(root)
# list the files
list.files(root, recursive = TRUE)

# remove all .tsv files from valid git2rdata objects
rm_data(root, path = ".")
# check the removal of the .tsv file
list.files(root, recursive = TRUE)
list_data(root)

# remove dangling git2rdata metadata files
prune_meta(root, path = ".")
# check the removal of the metadata
list.files(root, recursive = TRUE)
list_data(root)


## on git repo

# initialise a git repo using git2r
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# store a dataframe
write_vc(iris[1:6, ], "iris", repo, sorting = "Sepal.Length", stage = TRUE)
# check that the dataframe is stored
status(repo)
list_data(repo)

# commit the current version and check the git repo
commit(repo, "add iris data", session = TRUE)
status(repo)

# remove the data files from the repo
rm_data(repo, path = ".")
# check the removal
list_data(repo)
status(repo)

# remove dangling metadata
prune_meta(repo, path = ".")
# check the removal
list_data(repo)
status(repo)
```

---

prune_meta *Prune Metadata Files*

Description

Removes all **valid** metadata (`.yml` files) from the `path` when they don't have accompanying data (`.tsv` file). **Invalid** metadata triggers a warning without removing the metadata file.

Use this function with caution since it will remove all valid metadata files without asking for confirmation. We strongly recommend to use this function on files under version control. See `vignette("workflow", package = "git2rdata")` for some examples on how to use this.

Usage

```
prune_meta(root = ".", path = NULL, recursive = TRUE, ...)

## S3 method for class 'git_repository'
prune_meta(root, path = NULL, recursive = TRUE, ..., stage = FALSE)
```

Arguments

`root`	The root of a project. Can be a file path or a `git-repository`. Defaults to the current working directory (`"."`).
`path`	the directory in which to clean all the data files. The directory is relative to `root`.
`recursive`	remove files in subdirectories too.
`...`	parameters used in some methods
`stage`	stage the changes after removing the files. Defaults to `FALSE`.

Value

returns invisibly a vector of removed files names. The paths are relative to `root`.

See Also

Other storage: `list_data()`, `read_vc()`, `relabel()`, `rename_variable()`, `rm_data()`, `verify_vc()`, `write_vc()`

Examples

```
## on file system

# create a directory
root <- tempfile("git2rdata-")
dir.create(root)

# store a dataframe as git2rdata object. Capture the result to minimise
# screen output
junk <- write_vc(iris[1:6, ], "iris", root, sorting = "Sepal.Length")
# write a standard tab separate file (non git2rdata object)
write.table(iris, file = file.path(root, "standard.tsv"), sep = "\t")
# write a YAML file
yml <- list(
  authors = list(
   "Research Institute for Nature and Forest" = list(
```

```
       href = "https://www.inbo.be/en")))
yaml::write_yaml(yml, file = file.path(root, "_pkgdown.yml"))

# list the git2rdata objects
list_data(root)
# list the files
list.files(root, recursive = TRUE)

# remove all .tsv files from valid git2rdata objects
rm_data(root, path = ".")
# check the removal of the .tsv file
list.files(root, recursive = TRUE)
list_data(root)

# remove dangling git2rdata metadata files
prune_meta(root, path = ".")
# check the removal of the metadata
list.files(root, recursive = TRUE)
list_data(root)


## on git repo

# initialise a git repo using git2r
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# store a dataframe
write_vc(iris[1:6, ], "iris", repo, sorting = "Sepal.Length", stage = TRUE)
# check that the dataframe is stored
status(repo)
list_data(repo)

# commit the current version and check the git repo
commit(repo, "add iris data", session = TRUE)
status(repo)

# remove the data files from the repo
rm_data(repo, path = ".")
# check the removal
list_data(repo)
status(repo)

# remove dangling metadata
prune_meta(repo, path = ".")
# check the removal
list_data(repo)
status(repo)
```

pull *Re-exported Function From* git2r

Description

See pull in git2r.

See Also

Other version_control: commit(), push(), recent_commit(), repository(), status()

push *Re-exported Function From* git2r

Description

See push in git2r.

See Also

Other version_control: commit(), pull(), recent_commit(), repository(), status()

read_vc *Read a Git2rdata Object from Disk*

Description

read_vc() handles git2rdata objects stored by write_vc(). It reads and verifies the metadata file (.yml). Then it reads and verifies the raw data. The last step is back-transforming any transformation done by meta() to return the data.frame as stored by write_vc().

read_vc() is an S3 generic on root which currently handles "character" (a path) and "git-repository" (from git2r). S3 methods for other version control system could be added.

Usage

```
read_vc(file, root = ".")
```

Arguments

file	the name of the git2rdata object. Git2rdata objects cannot have dots in their name. The name may include a relative path. file is a path relative to the root. Note that file must point to a location within root.
root	The root of a project. Can be a file path or a git-repository. Defaults to the current working directory ("..").

Value

The `data.frame` with the file names and hashes as attributes.

See Also

Other storage: `list_data()`, `prune_meta()`, `relabel()`, `rename_variable()`, `rm_data()`, `verify_vc()`, `write_vc()`

Examples

```
## on file system

# create a directory
root <- tempfile("git2rdata-")
dir.create(root)

# write a dataframe to the directory
write_vc(iris[1:6, ], file = "iris", root = root, sorting = "Sepal.Length")
# check that a data file (.tsv) and a metadata file (.yml) exist.
list.files(root, recursive = TRUE)
# read the git2rdata object from the directory
read_vc("iris", root)

# store a new version with different observations but the same metadata
write_vc(iris[1:5, ], "iris", root)
list.files(root, recursive = TRUE)
# Removing a column requires version requires new metadata.
# Add strict = FALSE to override the existing metadata.
write_vc(
  iris[1:6, -2], "iris", root, sorting = "Sepal.Length", strict = FALSE
)
list.files(root, recursive = TRUE)
# storing the orignal version again requires another update of the metadata
write_vc(iris[1:6, ], "iris", root, sorting = "Sepal.Width", strict = FALSE)
list.files(root, recursive = TRUE)
# optimize = FALSE stores the data more verbose. This requires larger files.
write_vc(
  iris[1:6, ], "iris2", root, sorting = "Sepal.Width", optimize = FALSE
)
list.files(root, recursive = TRUE)



## on git repo using a git2r::git-repository

# initialise a git repo using the git2r package
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# store a dataframe in git repo.
```

```
write_vc(iris[1:6, ], file = "iris", root = repo, sorting = "Sepal.Length")
# This git2rdata object is not staged by default.
status(repo)
# read a dataframe from a git repo
read_vc("iris", repo)

# store a new version in the git repo and stage it in one go
write_vc(iris[1:5, ], "iris", repo, stage = TRUE)
status(repo)

# store a verbose version in a different gir2data object
write_vc(
  iris[1:6, ], "iris2", repo, sorting = "Sepal.Width", optimize = FALSE
)
status(repo)
```

`recent_commit` *Retrieve the Most Recent File Change*

Description

Retrieve the most recent commit that added or updated a file or git2rdata object. This does not imply that file still exists at the current HEAD as it ignores the deletion of files.

Use this information to document the current version of file or git2rdata object in an analysis. Since it refers to the most recent change of this file, it remains unchanged by committing changes to other files. You can also use it to track if data got updated, requiring an analysis to be rerun. See `vignette("workflow", package = "git2rdata")`.

Usage

```
recent_commit(file, root, data = FALSE)
```

Arguments

`file`	the name of the git2rdata object. Git2rdata objects cannot have dots in their name. The name may include a relative path. `file` is a path relative to the `root`. Note that `file` must point to a location within `root`.
`root`	The root of a project. Can be a file path or a `git-repository`.
`data`	does `file` refers to a data object (`TRUE`) or to a file (`FALSE`)? Defaults to `FALSE`.

Value

a `data.frame` with `commit`, `author` and `when` for the most recent commit that adds op updates the file.

See Also

Other version_control: `commit()`, `pull()`, `push()`, `repository()`, `status()`

Examples

```
# initialise a git repo using git2r
repo_path <- tempfile("git2rdata-repo")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# write and commit a first dataframe
# store the output of write_vc() minimize screen output
junk <- write_vc(iris[1:6, ], "iris", repo, sorting = "Sepal.Length",
                 stage = TRUE)
commit(repo, "important analysis", session = TRUE)
list.files(repo_path)
Sys.sleep(1.1) # required because git doesn't handle subsecond timings

# write and commit a second dataframe
junk <- write_vc(iris[7:12, ], "iris2", repo, sorting = "Sepal.Length",
                 stage = TRUE)
commit(repo, "important analysis", session = TRUE)
list.files(repo_path)
Sys.sleep(1.1) # required because git doesn't handle subsecond timings

# write and commit a new version of the first dataframe
junk <- write_vc(iris[7:12, ], "iris", repo, stage = TRUE)
list.files(repo_path)
commit(repo, "important analysis", session = TRUE)



# find out in which commit a file was last changed

# "iris.tsv" was last updated in the third commit
recent_commit("iris.tsv", repo)
# "iris.yml" was last updated in the first commit
recent_commit("iris.yml", repo)
# "iris2.yml" was last updated in the second commit
recent_commit("iris2.yml", repo)
# the git2rdata object "iris" was last updated in the third commit
recent_commit("iris", repo, data = TRUE)

# remove a dataframe and commit it to see what happens with deleted files
file.remove(file.path(repo_path, "iris.tsv"))
prune_meta(repo, ".")
commit(repo, message = "remove iris", all = TRUE, session = TRUE)
list.files(repo_path)

# still points to the third commit as this is the latest commit in which the
# data was present
recent_commit("iris", repo, data = TRUE)
```

relabel *Relabel Factor Levels by Updating the Metadata*

Description

Imagine the situation where we have a dataframe with a factor variable and we have stored it with `write_vc(optimize = TRUE)`. The raw data file contains the factor indices and the metadata contains the link between the factor index and the corresponding label. See `vignette("version_control", package = "git2rdata")`. In such a case, relabelling a factor can be fast and lightweight by updating the metadata.

Usage

```
relabel(file, root = ".", change)
```

Arguments

`file`	the name of the git2rdata object. Git2rdata objects cannot have dots in their name. The name may include a relative path. `file` is a path relative to the `root`. Note that `file` must point to a location within `root`.
`root`	The root of a project. Can be a file path or a `git-repository`. Defaults to the current working directory (`"."`).
`change`	either a `list` or a `data.frame`. In case of a `list` is a named `list` with named `vectors`. The names of list elements must match the names of the variables. The names of the vector elements must match the existing factor labels. The values represent the new factor labels. In case of a `data.frame` it needs to have the variables `factor` (name of the factor), `old` (the old) factor label and `new` (the new factor label). `relabel()` ignores all other columns.

Value

invisible `NULL`.

See Also

Other storage: `list_data()`, `prune_meta()`, `read_vc()`, `rename_variable()`, `rm_data()`, `verify_vc()`, `write_vc()`

Examples

```
# initialise a git repo using git2r
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a dataframe and store it as an optimized git2rdata object.
# Note that write_vc() uses optimization by default.
```

```
# Stage and commit the git2rdata object.
ds <- data.frame(
  a = c("a1", "a2"),
  b = c("b2", "b1"),
  stringsAsFactors = TRUE
)
junk <- write_vc(ds, "relabel", repo, sorting = "b", stage = TRUE)
cm <- commit(repo, "initial commit")
# check that the workspace is clean
status(repo)

# Define new labels as a list and apply them to the git2rdata object.
new_labels <- list(
  a = list(a2 = "a3")
)
relabel("relabel", repo, new_labels)
# check the changes
read_vc("relabel", repo)
# relabel() changed the metadata, not the raw data
status(repo)
git2r::add(repo, "relabel.*")
cm <- commit(repo, "relabel using a list")

# Define new labels as a dataframe and apply them to the git2rdata object
change <- data.frame(
  factor = c("a", "a", "b"),
  old = c("a3", "a1", "b2"),
  new = c("c2", "c1", "b3"),
  stringsAsFactors = TRUE
)
relabel("relabel", repo, change)
# check the changes
read_vc("relabel", repo)
# relabel() changed the metadata, not the raw data
status(repo)
```

`rename_variable` *Rename a Variable*

Description

The raw data file contains a header with the variable names. The metadata list the variable names and their type. Changing a variable name and overwriting the `git2rdata` object with result in an error. Because it will look like removing an existing variable and adding a new one. Overwriting the object with `strict = FALSE` potentially changes the order of the variables, leading to a large diff.

Usage

```
rename_variable(file, change, root = ".", ...)
```

```
## S3 method for class 'character'
rename_variable(file, change, root = ".", ...)

## Default S3 method:
rename_variable(file, change, root, ...)

## S3 method for class 'git_repository'
rename_variable(file, change, root, ..., stage = FALSE, force = FALSE)
```

Arguments

file	the name of the git2rdata object. Git2rdata objects cannot have dots in their name. The name may include a relative path. `file` is a path relative to the `root`. Note that `file` must point to a location within `root`.
change	A named vector with the old names as values and the new names as names.
root	The root of a project. Can be a file path or a `git-repository`. Defaults to the current working directory (`"."`).
...	parameters used in some methods
stage	Logical value indicating whether to stage the changes after writing the data. Defaults to `FALSE`.
force	Add ignored files. Default is FALSE.

Details

This function solves this by only updating the raw data header and the metadata.

Value

invisible `NULL`.

See Also

Other storage: `list_data()`, `prune_meta()`, `read_vc()`, `relabel()`, `rm_data()`, `verify_vc()`, `write_vc()`

Examples

```
# initialise a git repo using git2r
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# Create a dataframe and store it as an optimized git2rdata object.
# Note that write_vc() uses optimization by default.
# Stage and commit the git2rdata object.
ds <- data.frame(
  a = c("a1", "a2"),
```

```
  b = c("b2", "b1"),
  stringsAsFactors = TRUE
)
junk <- write_vc(ds, "rename", repo, sorting = "b", stage = TRUE)
cm <- commit(repo, "initial commit")
# check that the workspace is clean
status(repo)

# Define change.
change <- c(new_name = "a")
rename_variable(file = "rename", change = change, root = repo)
# check the changes
read_vc("rename", repo)
status(repo)
```

`repository` *Re-exported Function From* `git2r`

Description

See `repository` in `git2r`.

See Also

Other version_control: `commit()`, `pull()`, `push()`, `recent_commit()`, `status()`

`rm_data` *Remove Data Files From Git2rdata Objects*

Description

Remove the data (`.tsv`) file from all valid git2rdata objects at the `path`. The metadata remains untouched. A warning lists any git2rdata object with **invalid** metadata. The function keeps any `.tsv` file with invalid metadata or from non-git2rdata objects.

Use this function with caution since it will remove all valid data files without asking for confirmation. We strongly recommend to use this function on files under version control. See `vignette("workflow", package = "git2rdata")` for some examples on how to use this.

Usage

```
rm_data(root = ".", path = NULL, recursive = TRUE, ...)

## S3 method for class 'git_repository'
rm_data(
  root,
  path = NULL,
```

```
  recursive = TRUE,
  ...,
  stage = FALSE,
  type = c("unmodified", "modified", "ignored", "all")
)
```

Arguments

root	The root of a project. Can be a file path or a git-repository. Defaults to the current working directory ("..").
path	the directory in which to clean all the data files. The directory is relative to root.
recursive	remove files in subdirectories too.
...	parameters used in some methods
stage	stage the changes after removing the files. Defaults to FALSE.
type	Defines the classes of files to remove. unmodified are files in the git history and unchanged since the last commit. modified are files in the git history and changed since the last commit. ignored refers to file listed in a .gitignore file. Selecting modified will remove both unmodified and modified data files. Selecting ignored will remove unmodified, modified and ignored data files. all refers to all visible data files, including untracked files.

Value

returns invisibly a vector of removed files names. The paths are relative to root.

See Also

Other storage: list_data(), prune_meta(), read_vc(), relabel(), rename_variable(), verify_vc(), write_vc()

Examples

```
## on file system

# create a directory
root <- tempfile("git2rdata-")
dir.create(root)

# store a dataframe as git2rdata object. Capture the result to minimise
# screen output
junk <- write_vc(iris[1:6, ], "iris", root, sorting = "Sepal.Length")
# write a standard tab separate file (non git2rdata object)
write.table(iris, file = file.path(root, "standard.tsv"), sep = "\t")
# write a YAML file
yml <- list(
  authors = list(
   "Research Institute for Nature and Forest" = list(
       href = "https://www.inbo.be/en")))
yaml::write_yaml(yml, file = file.path(root, "_pkgdown.yml"))
```

```
# list the git2rdata objects
list_data(root)
# list the files
list.files(root, recursive = TRUE)

# remove all .tsv files from valid git2rdata objects
rm_data(root, path = ".")
# check the removal of the .tsv file
list.files(root, recursive = TRUE)
list_data(root)

# remove dangling git2rdata metadata files
prune_meta(root, path = ".")
# check the removal of the metadata
list.files(root, recursive = TRUE)
list_data(root)


## on git repo

# initialise a git repo using git2r
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# store a dataframe
write_vc(iris[1:6, ], "iris", repo, sorting = "Sepal.Length", stage = TRUE)
# check that the dataframe is stored
status(repo)
list_data(repo)

# commit the current version and check the git repo
commit(repo, "add iris data", session = TRUE)
status(repo)

# remove the data files from the repo
rm_data(repo, path = ".")
# check the removal
list_data(repo)
status(repo)

# remove dangling metadata
prune_meta(repo, path = ".")
# check the removal
list_data(repo)
status(repo)
```

`status` *Re-exported Function From* `git2r`

Description

See status in git2r.

See Also

Other version_control: commit(), pull(), push(), recent_commit(), repository()

---

verify_vc *Read a file an verify the presence of variables*

Description

Reads the file with read_vc(). Then verifies that every variable listed in variables is present in the data.frame.

Usage

```
verify_vc(file, root, variables)
```

Arguments

file	the name of the git2rdata object. Git2rdata objects cannot have dots in their name. The name may include a relative path. file is a path relative to the root. Note that file must point to a location within root.
root	The root of a project. Can be a file path or a git-repository. Defaults to the current working directory ("..").
variables	a character vector with variable names.

See Also

Other storage: list_data(), prune_meta(), read_vc(), relabel(), rename_variable(), rm_data(), write_vc()

---

write_vc *Store a Data.Frame as a Git2rdata Object on Disk*

Description

A git2rdata object consists of two files. The ".tsv" file contains the raw data as a plain text tab separated file. The ".yml" contains the metadata on the columns in plain text YAML format. See vignette("plain text", package = "git2rdata") for more details on the implementation.

Usage

```
write_vc(
  x,
  file,
  root = ".",
  sorting,
  strict = TRUE,
  optimize = TRUE,
  na = "NA",
  ...,
  split_by
)

## S3 method for class 'character'
write_vc(
  x,
  file,
  root = ".",
  sorting,
  strict = TRUE,
  optimize = TRUE,
  na = "NA",
  ...,
  split_by = character(0)
)

## S3 method for class 'git_repository'
write_vc(
  x,
  file,
  root,
  sorting,
  strict = TRUE,
  optimize = TRUE,
  na = "NA",
  ...,
  stage = FALSE,
  force = FALSE
)
```

Arguments

x	the data.frame.
file	the name of the git2rdata object. Git2rdata objects cannot have dots in their name. The name may include a relative path. file is a path relative to the root. Note that file must point to a location within root.
root	The root of a project. Can be a file path or a git-repository. Defaults to the current working directory ("..").

sorting	an optional vector of column names defining which columns to use for sorting x and in what order to use them. The default empty sorting yields a warning. Add sorting to avoid this warning. Strongly recommended in combination with version control. See vignette("efficiency", package = "git2rdata") for an illustration of the importance of sorting.
strict	What to do when the metadata changes. strict = FALSE overwrites the data and the metadata with a warning listing the changes, strict = TRUE returns an error and leaves the data and metadata as is. Defaults to TRUE.
optimize	If TRUE, recode the data to get smaller text files. If FALSE, meta() converts the data to character. Defaults to TRUE.
na	the string to use for missing values in the data.
...	parameters used in some methods
split_by	An optional vector of variables name to split the text files. This creates a separate file for every combination. We prepend these variables to the vector of sorting variables.
stage	Logical value indicating whether to stage the changes after writing the data. Defaults to FALSE.
force	Add ignored files. Default is FALSE.

Value

a named vector with the file paths relative to root. The names contain the hashes of the files.

Note

..generic is a reserved name for the metadata and is a forbidden column name in a data.frame.

See Also

Other storage: list_data(), prune_meta(), read_vc(), relabel(), rename_variable(), rm_data(), verify_vc()

Examples

```
## on file system

# create a directory
root <- tempfile("git2rdata-")
dir.create(root)

# write a dataframe to the directory
write_vc(iris[1:6, ], file = "iris", root = root, sorting = "Sepal.Length")
# check that a data file (.tsv) and a metadata file (.yml) exist.
list.files(root, recursive = TRUE)
# read the git2rdata object from the directory
read_vc("iris", root)

# store a new version with different observations but the same metadata
```

```
write_vc(iris[1:5, ], "iris", root)
list.files(root, recursive = TRUE)
# Removing a column requires version requires new metadata.
# Add strict = FALSE to override the existing metadata.
write_vc(
  iris[1:6, -2], "iris", root, sorting = "Sepal.Length", strict = FALSE
)
list.files(root, recursive = TRUE)
# storing the orignal version again requires another update of the metadata
write_vc(iris[1:6, ], "iris", root, sorting = "Sepal.Width", strict = FALSE)
list.files(root, recursive = TRUE)
# optimize = FALSE stores the data more verbose. This requires larger files.
write_vc(
  iris[1:6, ], "iris2", root, sorting = "Sepal.Width", optimize = FALSE
)
list.files(root, recursive = TRUE)



## on git repo using a git2r::git-repository

# initialise a git repo using the git2r package
repo_path <- tempfile("git2rdata-repo-")
dir.create(repo_path)
repo <- git2r::init(repo_path)
git2r::config(repo, user.name = "Alice", user.email = "alice@example.org")

# store a dataframe in git repo.
write_vc(iris[1:6, ], file = "iris", root = repo, sorting = "Sepal.Length")
# This git2rdata object is not staged by default.
status(repo)
# read a dataframe from a git repo
read_vc("iris", repo)

# store a new version in the git repo and stage it in one go
write_vc(iris[1:5, ], "iris", repo, stage = TRUE)
status(repo)

# store a verbose version in a different gir2data object
write_vc(
  iris[1:6, ], "iris2", repo, sorting = "Sepal.Width", optimize = FALSE
)
status(repo)
```

Index