Package 'vmTools'

July 25, 2025

Type Package

Title Version Management Tools on the File System

Version 1.0.1

Maintainer Sam Byrne <ssbyrne@uw.edu>

Description Data version management on the file system for smaller projects. Manage data pipeline outputs with symbolic folder links, structured logging and reports, using 'R6' classes for encapsulation and 'data.table' for speed. Directory-specific logs used as source of truth to allow portability of versioned data folders.

License MIT + file LICENSE

Encoding UTF-8

Imports R6 (>= 2.5.1), data.table (>= 1.10.0)

Suggests devtools (>= 2.4.5), knitr (>= 1.46), rmarkdown (>= 2.27), testthat (>= 3.0.0), withr (>= 2.5.0), qpdf

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

URL https://github.com/epi-sam/vmTools

BugReports https://github.com/epi-sam/vmTools/issues

NeedsCompilation no

Author Sam Byrne [aut, cre, cph] (ORCID: <https://orcid.org/0009-0008-1067-307X>)

Repository CRAN

Date/Publication 2025-07-25 16:50:02 UTC

Contents

`assert_dir_exists` *Assert a directory exists on disk*

Description

Assert a directory exists on disk

Usage

```
assert_dir_exists(x)
```

Arguments

x	[chr] A directory path

Value

[none] stop if assertion fails

See Also

Other assertions: `assert_named_list()`, `assert_scalar()`, `assert_scalar_not_empty()`, `assert_type()`

assert_named_list *Assert an object is a list with named elements*

Description

Stops if:

- x is not a list
- x is a data.frame
- x has no names
- x has any NA names
- x has any zero-length names
- x has any whitespace-only names

Usage

```
assert_named_list(x)
```

Arguments

x	[list] List to check

Value

[none] stop if assertion fails

See Also

Other assertions: `assert_dir_exists()`, `assert_scalar()`, `assert_scalar_not_empty()`, `assert_type()`

assert_scalar *Assert an element is atomic and length 1*

Description

Assert an element is atomic and length 1

Usage

```
assert_scalar(x)
```

Arguments

x	[any] Element to check

Value

[none] stop if assertion fails

See Also

Other assertions: `assert_dir_exists()`, `assert_named_list()`, `assert_scalar_not_empty()`, `assert_type()`

`assert_scalar_not_empty` *Assert x is a scalar, and not empty in some way*

Description

Assert x is a scalar, and not empty in some way

Usage

```
assert_scalar_not_empty(x)
```

Arguments

`x`	[any] some object to check

Value

[none] stop if assertion fails

See Also

Other assertions: `assert_dir_exists()`, `assert_named_list()`, `assert_scalar()`, `assert_type()`

`assert_type` *Assert an object is a scalar of a certain type*

Description

Assert an object is a scalar of a certain type

Usage

```
assert_type(x, type)
```

Arguments

`x`	[any] Object to check
`type`	[chr] Type to check against

Value

[none] stop if assertion fails

See Also

Other assertions: `assert_dir_exists()`, `assert_named_list()`, `assert_scalar()`, `assert_scalar_not_empty()`

---

clean_path *Wrapper utility for sanitizing file.path(...) output*

Description

Wrapper utility for sanitizing file.path(...) output

Usage

```
clean_path(..., normalize = TRUE, mustWork = FALSE)
```

Arguments

`...`	[chr] paths passed to file.path()
`normalize`	[lgl] pass path to normalizePath()?
`mustWork`	[lgl] passed to normalizePath()

Value

[chr] full file paths with consistent platform-specific structure

Examples

```
clean_path(tempdir(), "/some/other/path/") # build a single path like file.path
clean_path(c(".", tempdir(), "/some/other/path/")) # vectorized
```

---

dir_tree *Print a directory tree to stdout*

Description

Print a directory tree to stdout

Usage

```
dir_tree(path = ".", level = Inf, prefix = "")
```

Arguments

path	[chr] The path to the directory to print
level	[int] The maximum depth to print
prefix	[chr] The prefix to add to each line

---

find_latest_output_dir

Find the latest output directory with format YYYY_MM_DD.VV

Description

Used only for signaling/messaging

Usage

```
find_latest_output_dir(root)
```

Arguments

root	[chr] path to root of output results

Value

[chr] path to latest output directory

---

find_n_cores

Cross platform helper to find number of cores

Description

Cross platform helper to find number of cores

Usage

```
find_n_cores()
```

Value

[int]

get_latest_output_date_index

get the latest index for given an output dir and a date

Description

directories are assumed to be named in YYYY_MM_DD.VV format with sane year/month/date/version values.

Usage

```
get_latest_output_date_index(dir, date)
```

Arguments

dir	[chr] path to directory with versioned dirs
date	[chr] character in YYYY_MM_DD format

Value

[int] largest version in directory tree or 0 if there are no version OR the directory tree does not exist

get_new_version_name

Increment a new output folder version as "YYYY_MM_DD.VV"

Description

Return on the date-version, not the full path. Does not create a folder.

Usage

```
get_new_version_name(root, date = "today")
```

Arguments

root	[chr] path to root of output results
date	[chr] character date in form of "YYYY_MM_DD" or "today". "today" will be interpreted as today's date.

Value

[chr] new output version of the form "YYYY_MM_DD.VV"

Examples

```
get_new_version_name(root = tempdir(), date = "today") # expect "YYYY_MM_DD.01"
```

is_an_error *Determine if an object is an error*

Description

Determine if an object is an error

Usage

```
is_an_error(x)
```

Arguments

x	[obj] some R object

Value

[lgl] TRUE / FALSE

See Also

Other validations: `validate_dir_exists()`, `validate_not_empty()`

is_windows *Is the current OS windows*

Description

Is the current OS windows

Usage

```
is_windows()
```

Value

[lgl]

is_windows_admin *If running on windows, check if the user has admin privileges*

Description

If running on windows, check if the user has admin privileges

Usage

```
is_windows_admin()
```

Value

[lgl] TRUE if the user in on a windows OS and has admin privileges, FALSE otherwise

lapply_depth *lapply at some list dept*

Description

Very simple replacement for purrr::map_depth to remove package dependency, but not very robust. Internal package use only in select cases.

Usage

```
lapply_depth(.x, .depth, .f, ...)
```

Arguments

.x	[list] List to apply function to
.depth	[integer] Depth to apply function at
.f	[function] Function to apply
...	[any] Additional arguments to pass to .f

Value

[list] List with function applied at target depth

print.Symlink_Tool *Symlink Tool custom print method*

Description

Symlink Tool custom print method

Usage

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

Arguments

x	[Symlink_Tool] The SLT class
...	[any] Additional arguments to ‘print()‘

Value

[stdout]

Examples

```
SLT
```

SLT *SymlinkTool R6 class*

Description

Class for lightweight file-system level data versioning, logs and reports without need for a database.

Methods

Public methods:

- SLT$new()
- SLT$return_dictionaries()
- SLT$return_dynamic_fields()
- SLT$mark_best()
- SLT$mark_keep()
- SLT$mark_remove()
- SLT$unmark()
- SLT$roundup_best()
- SLT$roundup_keep()

- SLT$roundup_remove()
- SLT$roundup_unmarked()
- SLT$roundup_by_date()
- SLT$get_common_new_version_name()
- SLT$make_new_version_folder()
- SLT$make_new_log()
- SLT$delete_version_folders()
- SLT$make_reports()
- SLT$clone()

Method `new()`**:** Initialize the SymlinkTool object - an R6 class

The constructor function.

Usage:

```
SLT$new(
  user_root_list = NULL,
  user_central_log_root = NULL,
  schema_repair = TRUE,
  verbose = TRUE,
  verbose_startup = FALSE,
  csv_reader = "fread_quiet",
  timezone = Sys.timezone()
)
```

Arguments:

`user_root_list` [list] Named list of root directories for pipeline outputs. This is where 'version_name' folders live - these are iterative runs of an analysis pipeline.

`user_central_log_root` [path] Root directory for the central log. If you have multiple roots in the 'user_root_list', you probably want the central log to live one level above those roots.

`schema_repair` [logical] Default 'TRUE'. If 'TRUE', the tool will attempt to repair any schema mismatches it finds in the logs when reading and writing e.g. add new columns if the tool schema has columns that existing logs do not. If 'FALSE', the tool will stop and throw an error if it finds a schema mismatch.

`verbose` [lgl: default TRUE] control message verbosity - if TRUE, standard message, if FALSE, warn only if something is irregular.

`verbose_startup` [lgl] see start up warnings, if relevant?

`csv_reader` [chr] The CSV reader to use (also assigns matching CSV writer). CAUTION: DO NOT USE 'data.table::fread' if you have any quotation marks (") in log comments (these lead to exploding series of quotations). https://github.com/Rdatatable/data.table/issues/4779. Otherwise use 'read.csv[2]'. Options:

- fread_quiet - 'data.table::fread' and suppress warnings (default)
- fread - 'data.table::fread'
- read.csv - 'utils::read.csv' - safer
- read.csv2 - 'utils::read.csv2' - safer, comma as decimal point, semicolon as field separator

`timezone` [chr] Default 'America/Los_Angeles'. The timezone to use for datestamps in logs. Must be a valid 'OlsonNames()' string.

Returns: [symlink_tool] A symlink tool object. You can instantiate a.k.a. create multiple objects, each of which has different roots and central logs.

Examples:

```
try(SLT$new()) # call with no arguments to see instructions
# Tool will not instantiate on Windows unless running with Admin permissions
# - requirement for symlink creation on Windows
```

Method `return_dictionaries()`**:** Return the contents of all private dictionaries.

Usage:

```
SLT$return_dictionaries(item_names = NULL)
```

Arguments:

`item_names` [chr] Default 'NULL'. If 'NULL', show all static internal fields. Otherwise, vector of static field names you want to see.

Returns: [list] of all static internal fields

Method `return_dynamic_fields()`**:** Print the contents of all dynamic fields.

Usage:

```
SLT$return_dynamic_fields(item_names = NULL)
```

Arguments:

`item_names` [chr] Default 'NULL'. If 'NULL', show all dynamic internal fields. Otherwise, vector of dynamic field names you want to see.

Returns: [std_out] Print dynamic field values to std_out.

Method `mark_best()`**:** Mark an output folder with a "best" symlink.

Enforces: - maximum of one best model - does not go back through history to make a best model from a prior version (not capable, this is what log_tool is for)

Writes: - appends to a log file in the output folder with a date and time stamp - appends a line to the central log file with a date and time stamp

Usage:

```
SLT$mark_best(version_name, user_entry)
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the 'root's you define when you instantiate the tool.

`user_entry` [list] Named list of user-defined fields to append to the log. After making a tool called e.g. slt, call 'slt$return_dictionaries("log_fields_user")' to find which fields a user may add. If you want to make your own version of this class, you may update 'log_schema' in the 'private$DICT' section to allow for them.

Returns: [ste_err] Messages about actions taken.

Method `mark_keep()`**:** Mark an output folder with a "keep_<version_name>" symlink

Writes: - appends to a log file in the output folder with a date and time stamp - appends a line to the central log file with a date and time stamp

Usage:

```
SLT$mark_keep(version_name, user_entry)
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the 'root's you define when you instantiate the tool.

`user_entry` [list] Named list of user-defined fields to append to the log. After making a tool called e.g. slt, call 'slt$return_dictionaries("log_fields_user")' to find which fields a user may add. If you want to make your own version of this class, you may update 'log_schema' in the 'private$DICT' section to allow for them.

Returns: [std_err] Messages about actions taken.

Method `mark_remove()`**:** Mark an output folder with a "remove_<version_name>" symlink

Indication that the results can be deleted - In the future, this will be used to remove old versions of the output, and provide a list of ST-GPR models to delete

Writes: - appends to a log file in the output folder with a date and time stamp - appends a line to the central log file with a date and time stamp

Usage:

```
SLT$mark_remove(version_name, user_entry)
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the 'root's you define when you instantiate the tool.

`user_entry` [list] Named list of user-defined fields to append to the log. After making a tool called e.g. slt, call 'slt$return_dictionaries("log_fields_user")' to find which fields a user may add. If you want to make your own version of this class, you may update 'log_schema' in the 'private$DICT' section to allow for them.

Returns: [std_err] Messages about actions taken.

Method `unmark()`**:** Remove all symlinks for a single 'version_name' in all 'roots'

Writes: - appends to a log file in the output folder with a date and time stamp - does _not_ append to the central log file

Usage:

```
SLT$unmark(version_name, user_entry)
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the 'root's you define when you instantiate the tool.

`user_entry` [list] Named list of user-defined fields to append to the log. After making a tool called e.g. slt, call 'slt$return_dictionaries("log_fields_user")' to find which fields a user may add. If you want to make your own version of this class, you may update 'log_schema' in the 'private$DICT' section to allow for them.

Returns: [std_err] Messages about the symlinks removed.

Method `roundup_best()`**:** Find all 'best_' symlinks in all 'roots'

Return both the symlink and the resolved symlink (folder the symlink points to)

Usage:

```
SLT$roundup_best()
```

Returns: [list] list of data.tables - one for each ‘root‘

Method `roundup_keep()`**:** Find all ‘keep_‘ symlinks in all ‘roots‘

Return both the symlink and the resolved symlink (folder the symlink points to)

Usage:

```
SLT$roundup_keep()
```

Returns: [list] list of data.tables - one for each ‘root‘

Method `roundup_remove()`**:** Find all ‘remove_‘ symlinks in all ‘roots‘

Return both the symlink and the resolved symlink (folder the symlink points to)

Usage:

```
SLT$roundup_remove()
```

Returns: [list] list of data.tables - one for each ‘root‘

Method `roundup_unmarked()`**:** Find all folders without symlinks in all ‘roots‘

Useful if you’re rapidly iterating, have only marked a couple folders, and want to remove the rest.

Usage:

```
SLT$roundup_unmarked()
```

Returns: [list] list of data.tables - one for each ‘root‘

Method `roundup_by_date()`**:** Find all ‘version_name‘ folders by creation date

Only finds folders that _have a log_, and reads creation date on first row. User may select dates by (using the ‘date_selector‘ argument): - greater than - ‘gt‘ - greater than or equal to - ‘gte‘ - less than - ‘nt‘ - less than or equal to ‘nte‘ - equal to ‘e‘

Usage:

```
SLT$roundup_by_date(user_date, date_selector)
```

Arguments:

`user_date` [c("character", "Date", POSIXct", "POSIXt")] A date with class requirements - must be formatted "2020-01-01 or 2020_01_01 or 2020/01/01"

`date_selector` [chr] See docstring explanation.

Returns: [list] list of data.tables - one for each ‘root‘

Method `get_common_new_version_name()`**:** Get a new YYYY_MM_DD.VV version compatible with _ALL THE TOOL’S ROOTS_

If root1 has 2025_01_01.01 and root2 has 2025_01_01.03, then a new folder would need to be 2025_01_01.04

Usage:

```
SLT$get_common_new_version_name(date = "today", root_list = private$DICT$ROOTS)
```

Arguments:

`date` [chr] Default "today". The date to use for the new version name. Must be formatted "2020_01_01"

`root_list` [list] named list of root directories for pipeline

Returns: [chr] format YYYY_MM_DD.VV

Method `make_new_version_folder()`: Create a new ‘version_name‘ folder in _ALL THE TOOL’S ROOTS_

Create a new log in each folder. No symlinks are created. No ‘user_entry‘ is used.

Usage:

```
SLT$make_new_version_folder(version_name = self$get_common_new_version_name())
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the ‘root‘s you define when you instantiate the tool. For convenience, user may leave NULL (default) and ‘get_common_new_version_name()‘ is used on that root.

Returns: [std_err] Messages about the folder creation.

Method `make_new_log()`: Safely write an empty log file for first pipeline runs

When you start a new pipeline run, make an empty log - helpful if you let this tool manage all your versions - you can roundup version_names by creation date using the log’s first entry - the file system doesn’t track directory creation dates (at time of writing)

Usage:

```
SLT$make_new_log(version_name)
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the ‘root‘s you define when you instantiate the tool.

Returns: [std_err] Messages about the log creation.

Method `delete_version_folders()`: Delete a ‘version_name‘ folder marked with a ‘remove_‘ symlink from _ALL ITS ROOTS_

Removes the symlink(s) and the underlying folder(s), and updates central log if folders were removed.

Writes: - appends a line to the central log file with a date and time stamp

Usage:

```
SLT$delete_version_folders(version_name, user_entry, require_user_input = TRUE)
```

Arguments:

`version_name` [chr] The directory name of the output folder that lives directly under one of the ‘root‘s you define when you instantiate the tool.

`user_entry` [list] Named list of user-defined fields to append to the log. After making a tool called e.g. slt, call ‘slt$return_dictionaries("log_fields_user")‘ to find which fields a user may add. If you want to make your own version of this class, you may update ‘log_schema‘ in the ‘private$DICT‘ section to allow for them.

`require_user_input` [lgl] if ‘TRUE‘, will prompt user to confirm deletion.

Returns: [std_err] Messages about deletion events.

Method `make_reports()`: Make all reports

Writes all reports to a summary .csv for every ‘root‘ defined in the tool.

Usage:

```
SLT$make_reports()
```

Returns: [std_err] Messages about where reports were written.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SLT$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SLT$new`
## ------------------------------------------------


try(SLT$new()) # call with no arguments to see instructions
# Tool will not instantiate on Windows unless running with Admin permissions
# - requirement for symlink creation on Windows
```

`split_line_breaks` *Split a character vector by line breaks*

Description

Split a character vector by line breaks

Usage

```
split_line_breaks(string)
```

Arguments

`string`	[chr] A character vector.

Value

All elements of the character vector are split by ‘\n‘ into new elements.

validate_dir_exists *Validate whether a directory exists*

Description

Validate whether a directory exists

Usage

```
validate_dir_exists(x, verbose = TRUE)
```

Arguments

x	[path] A directory path
verbose	[lgl] message to std_out?

Value

[lgl] TRUE if directory exists, FALSE otherwise

See Also

Other validations: `is_an_error()`, `validate_not_empty()`

validate_not_empty *Validate an object is not length 0, empty, blank etc.*

Description

Designed to also catch missing args when called inside a function.

Usage

```
validate_not_empty(x)
```

Arguments

x	[any] some argument to check

Value

[lgl] FALSE if empty in some way, TRUE otherwise

See Also

Other validations: `is_an_error()`, `validate_dir_exists()`

Index