

# Package: vmTools (via r-universe)

May 24, 2026

**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** <https://cran.r-universe.dev>

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

**RemoteUrl** <https://github.com/cran/vmTools>

**RemoteRef** HEAD

**RemoteSha** bc14822bb1c6445584cb0a93eb5e61e3d0e7a1e9

## Contents

assert_dir_exists . . . . .	2
assert_named_list . . . . .	3
assert_scalar . . . . .	3
assert_scalar_not_empty . . . . .	4
assert_type . . . . .	4
clean_path . . . . .	5
dir_tree . . . . .	5
find_latest_output_dir . . . . .	6
find_n_cores . . . . .	6
get_latest_output_date_index . . . . .	7
get_new_version_name . . . . .	7
is_an_error . . . . .	8
is_windows . . . . .	8
is_windows_admin . . . . .	9
lapply_depth . . . . .	9
print.Symlink_Tool . . . . .	10
SLT . . . . .	10
split_line_breaks . . . . .	16
validate_dir_exists . . . . .	17
validate_not_empty . . . . .	17
<b>Index</b>	<b>18</b>

---

assert_dir_exists	<i>Assert a directory exists on disk</i>
-------------------	--

---

### 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	<i>Wrapper utility for sanitizing file.path(...) output</i>
------------	---

---

**Description**

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

**Usage**

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

**Arguments**

...	[chr] paths passed to file.path()
normalize	[lg] pass path to normalizePath()?
mustWork	[lg] 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	<i>Print a directory tree to stdout</i>
----------	---

---

**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**

<code>dir</code>	[chr] path to directory with versioned dirs
<code>date</code>	[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**

<code>root</code>	[chr] path to root of output results
<code>date</code>	[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	<i>If running on windows, check if the user has admin privileges</i>
------------------	--

---

**Description**

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

**Usage**

```
is_windows_admin()
```

**Value**

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

---

lapply_depth	<i>lapply at some list depth</i>
--------------	----------------------------------

---

**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**

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

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

`verbose_startup` [lg] 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` [lg] 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            [lg] message to std\_out?

**Value**

[lg] 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**

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

**See Also**

Other validations: [is\\_an\\_error\(\)](#), [validate\\_dir\\_exists\(\)](#)

# Index

## \* assertions

- assert\_dir\_exists, 2
- assert\_named\_list, 3
- assert\_scalar, 3
- assert\_scalar\_not\_empty, 4
- assert\_type, 4

## \* validations

- is\_an\_error, 8
- validate\_dir\_exists, 17
- validate\_not\_empty, 17

- assert\_dir\_exists, 2, 3–5
- assert\_named\_list, 2, 3, 4, 5
- assert\_scalar, 2, 3, 3–5
- assert\_scalar\_not\_empty, 2, 3, 4, 4, 5
- assert\_type, 2, 3, 4, 4

- clean\_path, 5

- dir\_tree, 5

- find\_latest\_output\_dir, 6
- find\_n\_cores, 6

- get\_latest\_output\_date\_index, 7
- get\_new\_version\_name, 7

- is\_an\_error, 8, 17
- is\_windows, 8
- is\_windows\_admin, 9

- lapply\_depth, 9

- print.Symlink\_Tool, 10

- SLT, 10
- split\_line\_breaks, 16

- validate\_dir\_exists, 8, 17, 17
- validate\_not\_empty, 8, 17, 17