Package: vmTools (via r-universe)

November 13, 2026

Type Package

Title Version Management Tools on the File System

Version 2.0.0

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

LazyData true

Imports R6 (>= 2.5.1), data.table (>= 1.10.0), checkmate (>= 2.0.0), lifecycle

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

VignetteBuilder knitr

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

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

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

Repository https://epi-sam.r-universe.dev

Date/Publication 2026-03-21 18:26:22 UTC

RemoteUrl https://github.com/epi-sam/vmtools

RemoteRef HEAD

RemoteSha 76474efff44fd5e2eacb96800ad08f3af9b79974

Contents

clean_path 2
get_new_version_name 3
get_symlink_registry 3
new_symlink_type 4
print.Symlink_Composer 5
print.Symlink_Tool 5
SLC 6
SLT 8

Index **17**

`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
```

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"
```

get_symlink_registry *Get the full symlink registry*

Description

Returns the named list of symlink types and their associated properties/functions. Expected entries are:

- is_singleton: (logical) whether this symlink type is a singleton (only one per root)
- symlink_suffix_fn: (function) function to create symlink name based on version name (for non-singleton symlink types)
- ls_regex_fn: (function) function to find a symlink (by version_name for non-singletons) when running ls system calls
- extract_regex: (function) function to get the regex pattern to find the symlink type

Usage

```
get_symlink_registry()
```

Value

(list) the full symlink registry as a named list

Examples

```
get_symlink_registry()
```

`new_symlink_type` *Register a new symlink type*

Description

Register a new symlink type

Usage

```
new_symlink_type(symlink_type, is_singleton, verbose = TRUE)
```

Arguments

`symlink_type`	(chr) name of the new symlink type to register. Must not be a canonical built-in symlink type `c("best", "keep", "remove", "latest")` Built-in symlink types are case-insensitive (i.e. the user may not register a "BEST" symlink type). User-defined symlink types may be overwritten.
`is_singleton`	(lgl) Can there be only one of this symlink type per root directory? If `TRUE`, the symlink name will be exactly `symlink_type` (e.g. "best"). If `FALSE`, the symlink name will be `symlink_type` + "_" + version_name (e.g. "keep_2025_01_01.01").
`verbose`	(lgl: default `TRUE`) control message verbosity when registering the new symlink type

Value

(invisible NULL) called for side-effect of registering the new symlink type in the package-level registry environment

Examples

```
new_symlink_type("stalled", is_singleton = FALSE)
```

print.Symlink_Composer *Symlink Composer custom print method*

Description

Symlink Composer custom print method

Usage

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

Arguments

x	(Symlink_Composer) The SLC class
...	(any) Additional arguments to print()

Value

(stdout)

Examples

```
SLC
```

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
```

SLC *SymlinkComposer R6 class*

Description

Manages multiple pre-instantiated SymlinkTool instances for aggregated query operations across multiple projects.

Methods

Public methods:

- `SLC$new()`
- `SLC$get_tool()`
- `SLC$list_tools()`
- `SLC$roundup_all()`
- `SLC$get_common_symlink_types()`
- `SLC$roundup_all_unmarked()`
- `SLC$roundup_all_by_date()`
- `SLC$clone()`

Method `new()`**:** Initialize the SymlinkComposer

Usage:

```
SLC$new(tool_list, verbose = FALSE)
```

Arguments:

`tool_list` (named list) Named list of pre-instantiated SLT objects. Names become tool identifiers for combined queries. Example: list(project_a = my_slt_a, project_b = my_slt_b)

`verbose` (lgl) Control message verbosity

Returns: (SymlinkComposer) A composer managing multiple SLT instances

Method `get_tool()`**:** Access individual SLT instances

Usage:

```
SLC$get_tool(tool_name)
```

Arguments:

`tool_name` (chr) Name of the tool to retrieve

Returns: (SLT) The requested SymlinkTool instance

Method `list_tools()`**:** List all tool names

Usage:

```
SLC$list_tools()
```

Returns: (chr) Vector of tool names

Method `roundup_all()`**:** Roundup symlinks of a given type across all tools

Valid types are determined by the intersection of `symlink_types` from all SLT instances held by this composer. Use `$get_common_symlink_types()` to inspect the valid set.

Usage:

```
SLC$roundup_all(symlink_type, return_as = c("table", "version_name"))
```

Arguments:

`symlink_type` (chr) One of the common symlink types across all tools, e.g. "best", "keep", "remove", "latest"

`return_as` (chr) `"table"` (default) or `"version_name"`

Returns: (data.table/chr)

Method `get_common_symlink_types()`**:** Return the symlink types common to all held SLT instances

These are the valid values for the `symlink_type` argument of `roundup_all()`.

Usage:

```
SLC$get_common_symlink_types()
```

Returns: (chr) Vector of common symlink type strings

Method `roundup_all_unmarked()`**:** Roundup all unmarked folders across all tools

Usage:

```
SLC$roundup_all_unmarked(return_as = c("table", "version_name"))
```

Arguments:

`return_as` (chr default: `"table"`) or `"version_name"`

Returns: (data.table) Combined results with 'tool_name' column

Method `roundup_all_by_date()`**:** Roundup folders by date across all tools

Usage:

```
SLC$roundup_all_by_date(
  user_date,
  date_selector,
  return_as = c("table", "version_name")
)
```

Arguments:

`user_date` (chr/Date/POSIXct) Date to query

`date_selector` (chr) One of: 'gt', 'gte', 'lt', 'lte', 'e'

`return_as` (chr default: `"table"`) or `"version_name"`

Returns: (data.table) Combined results with 'tool_name' column

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

SLT *SymlinkTool R6 class*

Description

Class for lightweight file-system level data versioning, logs and reports without need for a database. Manages four symlink types: 'best' (validated production), 'latest' (most recent run), 'keep' (archive-worthy), and 'remove' (marked for deletion).

Methods

Public methods:

- SLT$new()
- SLT$return_dictionaries()
- SLT$return_dynamic_fields()
- SLT$mark()
- SLT$mark_best()
- SLT$mark_latest()
- SLT$mark_keep()
- SLT$mark_remove()
- SLT$unmark()
- SLT$roundup()
- SLT$roundup_best()
- SLT$roundup_latest()
- 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(
  root_list = NULL,
  central_log_root = NULL,
  schema_repair = TRUE,
  verbose = FALSE,
```

```
  verbose_startup = FALSE,
  csv_reader = "fread_quiet",
  timezone = Sys.timezone()
)
```

Arguments:

`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.

`central_log_root` (path) Root directory for the central log. If you have multiple roots in the `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()`**:** Mark a version_name folder with a symlink of the specified type.

Generic dispatch method for all symlink marking. Legacy per-type methods (`mark_best`, `mark_latest`, etc.) are superceded thin wrappers around this.

Usage:

```
SLT$mark(symlink_type, version_name, user_entry)
```

Arguments:

`symlink_type` (chr) A registered symlink type, e.g. `"best"`, `"keep"`, or any type added via `new_symlink_type()`.

`version_name` (chr) The version_name folder to mark.

`user_entry` (list) Named list of user log fields.

Returns: (std_err) Messages about actions taken.

Method `mark_best()`**: [Superseded]**

Please use `mark("best", version_name, user_entry)` instead.

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 `roots` 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_latest()`**: [Superseded]**

Mark an output folder with a "latest" symlink.

Please use `mark("latest", version_name, user_entry)` instead.

Creates a 'latest' symlink (singleton) in each root. Auto-demotes current 'latest' if different. A folder cannot be both 'latest' and 'best' - this will auto-demote from the conflicting state.

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_latest(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_keep()`**: [Superseded]**

Mark an output folder with a "keep_<version_name>" symlink

Please use `mark("keep", version_name, user_entry)` instead.

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()`**: [Superseded]**

Mark an output folder with a "remove_<version_name>" symlink

Please use `mark("remove", version_name, user_entry)` instead.

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 `roots` 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 `roots` 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()`**:** Find all symlinks of a given type across all roots.

Generic dispatch method for all roundup operations. Legacy per-type methods (`roundup_best`, `roundup_keep`, etc.) are superceded thin wrappers.

Usage:

```
SLT$roundup(symlink_type, return_as = c("table", "version_name"))
```

Arguments:

`symlink_type` (chr) A registered symlink type.

`return_as` (chr: default `"table"`) or `"version_name"`.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as = version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

Method `roundup_best()`**: [Superseded]**

Find all `best` symlinks in all `roots`

Please use `roundup("best", return_as)` instead.

Usage:

```
SLT$roundup_best(return_as = c("table", "version_name"))
```

Arguments:

`return_as` (chr) Default `table`. Whether to return a data.table with symlink and resolved paths (`table`) or just a vector of `version_names` (`version_name`) for easier automated round-ups.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as` = `version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

Method `roundup_latest()`**: [Superseded]**

Find all `latest` symlinks in all `roots`

Usage:

```
SLT$roundup_latest(return_as = c("table", "version_name"))
```

Arguments:

`return_as` (chr) Default `table`. Whether to return a data.table with symlink and resolved paths (`table`) or just a vector of `version_names` (`version_name`) for easier automated round-ups.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as` = `version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

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

Usage:

```
SLT$roundup_keep(return_as = c("table", "version_name"))
```

Arguments:

`return_as` (chr) Default `table`. Whether to return a data.table with symlink and resolved paths (`table`) or just a vector of `version_names` (`version_name`) for easier automated round-ups.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as` = `version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

Method `roundup_remove()`**: [Superseded]**

Find all `remove_` symlinks in all `roots`

Usage:

```
SLT$roundup_remove(return_as = c("table", "version_name"))
```

Arguments:

`return_as` (chr) Default `table`. Whether to return a data.table with symlink and resolved paths (`table`) or just a vector of `version_names` (`version_name`) for easier automated round-ups.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as` = `version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

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

Usage:

```
SLT$roundup_unmarked(return_as = c("table", "version_name"))
```

Arguments:

`return_as` (chr) Default `table`. Whether to return a data.table with symlink and resolved paths (`table`) or just a vector of `version_names` (`version_name`) for easier automated round-ups.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as` = `version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

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,
  return_as = c("table", "version_name")
)
```

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.

`return_as` (chr) Default `table`. Whether to return a data.table with symlink and resolved paths (`table`) or just a vector of `version_names` (`version_name`) for easier automated round-ups.

Returns: (data.table/chr) A single data.table with results from all roots, with `root_name` as the first column, or a vector of `version_names` if `return_as` = `version_name`.
Table output schema

```
root_name version_name dir_name dir_name_resolved
   <char>       <char>   <char>            <char>
```

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 `roots` 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 `roots` 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 `roots` 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
```

Index

clean_path, 2

get_new_version_name, 3
get_symlink_registry, 3

new_symlink_type, 4

print.Symlink_Composer, 5
print.Symlink_Tool, 5

SLC, 6
SLT, 8