Package 'devkit'

June 19, 2026

Type Package

Title Practical Utilities for Package Development and Session Auditing

Version 1.0.0

Description Provides a comprehensive collection of practical R utilities designed to streamline the full life cycle of package development, interactive data analysis, and session management. It offers tools for package management, development workflows, debugging, data processing, and system optimization. These utilities help R developers and data scientists automate release tasks, manage and scan dependencies, audit session states, optimize memory usage, and scramble sensitive Personally Identifiable Information ('PII').

Encoding UTF-8

License MIT + file LICENSE

Language en-US

Depends R (>= 3.5.0)

Imports tools

Suggests testthat, devtools, roxygen2, usethis, knitr, rmarkdown, withr

VignetteBuilder knitr

RoxygenNote 8.0.0

Config/roxygen2/version 8.0.0

URL https://zankrut20.github.io/devkit/, https://github.com/zankrut20/devkit

BugReports https://github.com/zankrut20/devkit/issues

NeedsCompilation no

Author Zankrut Goyani [aut, cre, cph]

Maintainer Zankrut Goyani <zankrut20@gmail.com>

Repository CRAN

Date/Publication 2026-06-19 11:30:02 UTC

Contents

`architect_release` *Interactive Release Candidate Architect*

Description

Automates the process of preparing a package release by bumping the version in the 'DESCRIPTION' file and interactively drafting release notes in 'NEWS.md'.

Usage

```
architect_release()
```

Details

The function performs the following steps:

1. Verifies the existence of the ‘DESCRIPTION‘ file.
2. Parses the current version and prompts the user to choose between a Patch, Minor, or Major bump.
3. Updates the ‘Version‘ and ‘Date‘ fields in the ‘DESCRIPTION‘ file upon confirmation.
4. Interactively collects changelog items from the user and prepends them to ‘NEWS.md‘.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `package`, `old_version`, `new_version`, `bump_type`, `description_updated` (logical), and `news_updated` (logical).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  architect_release()
}
```

`architect_vignette` *Interactive Vignette Architect*

Description

Scaffolds a CRAN-compliant RMarkdown vignette by interactively prompting the user for metadata, narrative structure, and target functions to highlight.

Usage

```
architect_vignette()
```

Details

The function guides the user through the following process:

1. Collects package name and vignette title.
2. Prompts for a structural template (Quick Start, Deep Dive, or Case Study).
3. Requests a list of core functions to be featured in the vignette.
4. Generates a ‘.Rmd‘ file with a proper YAML header and a narrative scaffold based on the selected template.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `file_path`, `package`, `title`, `template`, and `functions` (character vector).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  architect_vignette()
}
```

`audit_dependencies` *Interactive Dependency Diplomat (Base R Edition)*

Description

Scans package source code for external package calls and cross-references them against the 'DESCRIPTION' file to interactively resolve missing, unused, or misclassified dependencies without requiring any external tools.

Usage

```
audit_dependencies()
```

Details

The function implements a zero-dependency approach to dependency auditing:

1. Parses the 'DESCRIPTION' file to identify currently declared 'Imports' and 'Suggests'.
2. Recursively scans the 'R/', 'tests/', and 'vignettes/' directories for '::' calls, 'library()' calls, and 'require()' calls.
3. Filters out base R packages.
4. Interactively prompts the user to add missing dependencies or remove unused ones from the 'DESCRIPTION' file.

Value

Invisibly returns a named list with components: `status` ("done", "clean", or "error"), `ghost_deps`, `bloat_deps`, `misclassified`, and `description_modified` (logical).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  audit_dependencies()
}
```

`audit_script` *Silent State Auditor*

Description

Takes a snapshot of global options, graphical parameters, and the working directory, sources a specified R script, and then interactively helps the user identify and revert any "hijacked" settings changed by the script.

Usage

```
audit_script()
```

Details

The function operates as follows:

1. Captures the current state of ‘getwd()‘, ‘options()‘, and ‘par()‘.
2. Prompts the user to select an R script from the current directory to execute.
3. Sources the selected script, catching any errors that occur during execution.
4. Captures the "Before" state (working directory, loaded packages, global objects).
5. Executes the target script.
6. Captures the "After" state and reports the delta.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `script`, and `changes_found` (logical).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  audit_script()
}
```

`benchmark_branches` *Branch-Based Performance Benchmarker*

Description

Interactively selects Git branches, runs a target R script in an isolated environment for each branch, and compares the execution times to identify performance regressions or improvements across different versions of the codebase.

Usage

```
benchmark_branches()
```

Details

The function performs the following workflow:

- Verifies that the current directory is a Git repository.
- Extracts available branches and allows the user to select specific ones or benchmark all.
- Prompts for the path to the R script to be benchmarked.
- Stashes any uncommitted changes to ensure a clean state.
- Iterates through the selected branches, checking each one out and executing the target script within a fresh environment `new.env()`.
- Restores the original branch and pops the stash to return the workspace to its initial state.

Value

A data frame containing the benchmark results, including the branch name, execution time in seconds, and the execution status (Success/Failed).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  benchmark_branches()
}
```

`bootstrap_dev_env` *Initialize Development Environment*

Description

Scans the system for core R package development tools, prompts the user to install any missing packages, and allows for the selective loading of these tools into the current R session.

Usage

```
bootstrap_dev_env()
```

Details

The function focuses on a standard toolkit for CRAN-ready development, including ‘devtools‘, ‘roxygen2‘, ‘usethis‘, ‘testthat‘, and ‘knitr‘.

The process follows these steps:

1. Compares the list of core tools against the currently installed packages.
2. If tools are missing, it provides an interactive menu to install all missing packages, specific ones, or skip installation entirely.
3. After ensuring availability, it prompts the user to select which of the available tools should be attached to the current session using ‘library()‘.

Value

Invisibly returns a named list with components: `status` ("done"), `initially_missing`, `available`, and `loaded` (character vectors).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  bootstrap_dev_env()
}
```

`detect_masking` *Function Masking Detective*

Description

Scans for namespace conflicts among attached packages and interactively helps the user resolve them. The function adapts its recommendations based on whether it detects a package development environment or a standalone analysis script.

Usage

```
detect_masking()
```

Details

The function performs the following steps:

1. Identifies all functions that are masked by multiple attached packages.
2. Interactively prompts the user to select the preferred package for each conflicting function.
3. If a `DESCRIPTION` file is present (Package Development context), it suggests updating the `Imports` field and adding `@importFrom` roxygen2 tags.
4. If no `DESCRIPTION` file is present (Standalone context), it generates a code snippet to explicitly assign the preferred functions and offers to apply them.

Value

Invisibly returns a named list with components: `status` ("done" or "clean"), `conflicts` (named list), `resolutions` (named list), and `context` ("package" or "standalone").

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  detect_masking()
}
```

`dictate_dictionary` *Interactive Data Dictionary Dictator*

Description

Generates roxygen2 documentation for a data frame by interactively prompting the user for a dataset title, a general description, and individual column descriptions.

Usage

```
dictate_dictionary()
```

Details

The function performs the following steps:

1. Scans the global environment for all available data frames.
2. Prompts the user to select a target data frame for documentation.
3. Collects high-level metadata (title and description) for the dataset.
4. Iterates through each column, displaying its type and prompting the user for a descriptive label.
5. Assembles the collected information into a CRAN-compliant roxygen2 block using the '\describe' tag.
6. Offers the user the choice to either print the resulting block to the console or append it to 'R/data.R'.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `dataset`, `dimensions`, `output`, and `roxygen_block` (character vector).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  dictate_dictionary()
}
```

`dispatch_checkpoints` *Checkpoint Dispatcher*

Description

A crash-resilient wrapper for batch processing that silently caches progress at defined intervals and interactively recovers from interrupted sessions.

Usage

```
dispatch_checkpoints(
  items,
  target_func,
  checkpoint_file = "batch_checkpoint.rds"
)
```

Arguments

`items`	A vector or list of items to process.
`target_func`	The function to apply to each item.
`checkpoint_file`	Character. The file path for the state cache. Defaults to ‘"batch_checkpoint.rds"‘.

Details

The function provides a safety layer for long-running batch operations:

1. Checks for an existing ‘.rds‘ checkpoint file. If found, it prompts the user to resume from the last saved index or restart the process.
2. Prompts the user for a save frequency (e.g., every 100 items).
3. Executes the ‘target_func‘ on each item in ‘items‘ within a ‘tryCatch‘ block.
4. If a critical error occurs, it immediately saves the current state to the ‘checkpoint_file‘ and halts execution.
5. Periodically saves the state based on the specified frequency.
6. Upon successful completion of all items, the temporary checkpoint file is deleted.

Value

A list of successfully processed results, or an invisible list with `status = "cancelled"` if aborted during resumption.

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  dispatch_checkpoints(items = list(), target_func = print)
}
```

`export_snapshot` *Export Session Snapshot*

Description

Generates a reproducible installation script for all currently attached external R packages, allowing the user to recreate the exact environment in a different session or on another machine.

Usage

```
export_snapshot()
```

Details

The function provides two modes of environment capture:

1. **Flexible Installation**: Generates a script that checks for missing packages and installs the latest available versions.
2. **Strict Version Locking**: Uses ‘devtools::install_version()‘ to lock the environment to the exact versions currently installed on the system, ensuring maximum reproducibility.

Value

Invisibly returns a named list with components: `status` ("done" or "clean"), `packages` (character vector), `file` (output path), and `version_locked` (logical).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  export_snapshot()
}
```

`hunt_zombies` *Zombie Data Hunter*

Description

Hunts down "invisible" memory hogs such as massive temporary files, unclosed graphical devices, and uncollected garbage, prompting the user for targeted cleaning to reclaim system resources.

Usage

```
hunt_zombies()
```

Details

The function performs a system scan for three types of "zombie" data:

1. **Temporary Files**: Calculates the total size of the current session's temporary directory. If it exceeds 5 MB, it offers to flush it.
2. **Graphical Devices**: Checks for any open graphical devices (e.g., PDF, PNG, or RStudioGD) and offers to close all of them.
3. **Orphaned Memory**: Offers to perform a "deep" garbage collection by calling ‘gc()‘ twice, which forces R to clear both older and newer generations of memory.

Value

Invisibly returns a named list with components: `status` ("done" or "cancelled"), `actions_taken` (character vector), `temp_flushed_mb` (numeric), and `devices_closed` (integer).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  hunt_zombies()
}
```

loop_guardian *Bulk Loop Guardian*

Description

A memory-safe wrapper for heavy iterations that monitors RAM usage and triggers an interactive failsafe if the environment approaches critical capacity.

Usage

```
loop_guardian(
  items,
  target_func,
  limit_mb = 4000,
  save_path = "emergency_checkpoint.rds"
)
```

Arguments

items	A vector or list of items to process.
target_func	The function to apply to each item.
limit_mb	Numeric. The memory limit in megabytes before triggering the alarm. Defaults to ‘4000‘.
save_path	Character. Where to dump the emergency checkpoint data. Defaults to ‘"emergency_checkpoint.rds"‘.

Details

The function provides a safeguard against memory-related crashes during large-scale batch processing:

1. Iterates through the ‘items‘ vector, applying ‘target_func‘ to each element.
2. Every 50 iterations, it checks the current memory usage using ‘gc()‘.
3. If the memory usage exceeds ‘limit_mb‘, it first attempts a deep garbage collection to recover RAM.
4. If memory remains above the threshold after GC, it triggers an interactive alarm, allowing the user to:
   - Save current progress to ‘save_path‘ and abort.
   - Ignore the limit and attempt to continue.
   - Abort immediately without saving.

Value

A list of successfully processed results.

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  loop_guardian(items = 1:5, target_func = print)
}
```

---

manage_deprecation *Interactive Lifecycle Manager*

Description

Scaffolds CRAN-compliant deprecation wrappers and interactively refactors old function calls across tests and vignettes to ensure a smooth transition to new API versions.

Usage

```
manage_deprecation()
```

Details

The function manages the deprecation lifecycle through the following steps:

1. Prompts the user for the name of the function to be deprecated and its replacement.
2. Automatically generates a deprecated wrapper function that calls '.Deprecated()' and then forwards arguments to the new function.
3. Appends this wrapper to 'R/deprecated.R', ensuring the package remains backward compatible while warning users.
4. Optionally scans the 'tests/' and 'vignettes/' directories for occurrences of the old function and interactively replaces them with the new one.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `old_function`, `new_function`, `wrapper_file`, and `replacements` (integer count).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  manage_deprecation()
}
```

`mask_identity` *Interactive Identity Masker*

Description

Safely anonymizes Personally Identifiable Information ('PII') in a dataset by interactively prompting the user to keep, drop, or scramble each column.

Usage

```
mask_identity(envir = parent.frame())
```

Arguments

`envir`	The environment to search for data frames and in which to save the anonymized dataset. Defaults to the calling environment.

Details

The function provides a guided workflow for data anonymization:

1. Scans the calling environment for available data frames and prompts the user to select one.
2. Iterates through every column in the selected data frame, displaying its name and type.
3. For each column, the user chooses one of three actions:
   - **Keep**: Leaves the column unchanged.
   - **Scramble**: For numeric data, it shuffles the values to preserve the distribution while breaking the link to individuals. For text/factors, it replaces values with sequential placeholders (e.g., "Masked_0001").
   - **Drop**: Removes the column entirely from the dataset.
4. Saves the resulting anonymized data frame back to `envir` with a `_masked` suffix.
5. Optionally generates a `dput()` output of the first 20 rows for easy, safe sharing.

Value

Invisibly returns the anonymized data frame.

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  mask_identity()
}
```

`network_diplomat` *The Network Diplomat*

Description

Safely executes network requests across a list of URLs or endpoints, ensuring server politeness through interactive rate limiting and resilience via automatic retries with exponential backoff.

Usage

```
network_diplomat(targets, target_func, max_retries = 3)
```

Arguments

`targets`	A vector of URLs or target IDs to process.
`target_func`	The function that makes the network request (takes one target).
`max_retries`	Integer. Maximum number of times to retry a single failure. Defaults to ‘3‘.

Details

The function implements a robust network request manager:

1. **Rate Limiting**: Prompts the user for the server's requests-per-minute limit and calculates a precise sleep interval between requests to avoid being blocked.
2. **Retry Logic**: Wraps each request in a ‘tryCatch‘ block. If a request fails, it will retry up to ‘max_retries‘ times.
3. **Exponential Backoff**: After each failure, the function waits for an increasing amount of time (5s, 10s, 20s, etc.) before retrying.
4. **HTTP 429 Handling**: If a "Too Many Requests" (HTTP 429) error is detected, it adds an additional penalty delay to the backoff time.
5. **Graceful Failure**: If all retries are exhausted, the target is marked as ‘NA‘ and the process continues to the next target.

Value

A list of successfully processed results, with ‘NA‘ for permanent failures.

Examples

```
if (interactive()) {
  network_diplomat(targets = c('https://example.com'), target_func = function(x) x)
}
```

`remove_package` *Interactive Package Remover*

Description

Safely removes a specified R package and its unused dependencies from the system, ensuring that other installed packages do not lose required dependencies.

Usage

```
remove_package(pkg, recursive = FALSE)
```

Arguments

`pkg`	Character. The name of the package to remove.
`recursive`	Logical. Whether to check for recursive dependencies. Defaults to 'FALSE'.

Details

The function implements a dependency-aware removal process:

1. Identifies the dependencies of the target package using 'tools::package_dependencies'.
2. Analyzes all other installed packages to determine which dependencies are still required.
3. Isolates "orphan" dependencies—those that were only required by the target package.
4. Interactively prompts the user to select which of these orphan packages (and the target package itself) should be removed.
5. Executes 'remove.packages()' on the selected items.

Value

A character vector of the packages that were removed, or invisibly 'character()' if nothing was removed.

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  remove_package('curl')
}
```

`remove_user_installed_packages` *Remove User-Installed Packages*

Description

Identifies and removes all user-installed R packages from the system, while carefully preserving base and recommended packages, as well as packages installed in specific system libraries (e.g., MRO).

Usage

```
remove_user_installed_packages()
```

Details

The function performs the following steps:

1. Retrieves a list of all installed packages.
2. Filters out packages located in libraries containing "MRO" to avoid corrupting system-specific installations.
3. Filters out packages with a priority of "base" or "recommended" to ensure core R functionality remains intact.
4. Identifies the library paths where the remaining user packages are installed.
5. Iteratively removes each identified user package using ‘remove.packages()‘.

Value

Invisibly returns a named list with components: `status` ("done" or "clean") and `packages_removed` (character vector).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  remove_user_installed_packages()
}
```

`scaffold_parallel` *Interactive CPU Architect*

Description

Scans local hardware to determine available CPU cores and generates a robust parallel processing scaffold for high-volume data tasks, automating the setup of a local cluster.

Usage

```
scaffold_parallel()
```

Details

The function guides the user through the creation of a parallel processing pipeline:

1. Detects the total number of available CPU cores using ‘parallel::detectCores()‘.
2. Prompts the user to select the number of cores to dedicate to the task, recommending leaving at least one core free for OS stability.
3. Collects the names of the target data object and the processing function.
4. Generates a complete R code snippet that:
   - Initializes a cluster using ‘makeCluster()‘.
   - Exports the required function to the worker nodes via ‘clusterExport()‘.
   - Executes the computation using ‘parLapply()‘.
   - Safely shuts down the cluster using ‘stopCluster()‘.
5. Offers to print the snippet to the console or save it directly to ‘parallel_scaffold.R‘.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `cores` (integer), `data_object`, `function_name`, and `saved_to` (file path or `NULL`).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  scaffold_parallel()
}
```

`scaffold_tests` *Interactive Test-Suite Architect*

Description

Scaffolds CRAN-compliant ‘testthat‘ boilerplate for a specific function by interactively prompting the user about expected behaviors, output types, and edge cases.

Usage

```
scaffold_tests()
```

Details

The function automates the creation of a test file in the ‘tests/testthat/‘ directory:

1. Verifies that the ‘tests/testthat‘ directory exists (suggesting ‘usethis::use_testthat()‘ if it does not).
2. Prompts for the name of the function to be tested and creates a corresponding ‘test-functionname.R‘ file.
3. Interactively determines the expected output type (e.g., data frame, list, numeric) to generate appropriate ‘expect_*‘ calls.
4. Asks whether to include tests for output dimensions or error handling for invalid inputs.
5. Writes a structured boilerplate file containing ‘test_that‘ blocks with TODO comments for the user to fill in mock data.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `function_name`, `test_file`, and `output_type`.

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  scaffold_tests()
}
```

`scan_dependencies` *Unused Dependency Scanner*

Description

Scans a specified R script for function calls, cross-references them against currently attached packages, and interactively helps the user resolve unused dependencies based on their current workflow.

Usage

```
scan_dependencies()
```

Details

The function provides different resolution paths depending on the detected context:

1. **Package Development**: If a 'DESCRIPTION' file is found, it flags unused packages that should be removed from the 'Imports' field.
2. **Data Analysis**: If the user is working in an active session, it offers to detach idle packages and call 'gc()' to reclaim RAM.
3. **Raw Script Cleaning**: It generates an optimized block of 'library()' calls containing only the packages actually required by the script.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", "clean", or "error"), `script`, `external_packages`, and `unused_packages` (character vectors).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  scan_dependencies()
}
```

`setup_preflight` *Interactive Pre-Flight Dispatcher*

Description

Generates a custom Git pre-commit hook by interactively prompting the user to select which safety checks should be enforced before allowing a commit.

Usage

```
setup_preflight()
```

Details

The function automates the creation of a '.git/hooks/pre-commit' shell script that can enforce the following checks:

1. **Code Styling**: Automatically runs 'styler::style_pkg()' to standardize formatting and stages the modified files.
2. **Documentation**: Automatically runs 'devtools::document()' to ensure the 'NAMESPACE' and help files are up to date.
3. **Testing**: Executes 'testthat::test_local()' and aborts the commit if any tests fail.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `hook_path`, and `checks` (named logical list).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  setup_preflight()
}
```

`setup_sentinel` *Session Sentinel*

Description

Interactively configures dual-logging for the current R session, routing messages, warnings, and errors to a text file while maintaining live console output.

Usage

```
setup_sentinel()
```

Details

The function implements a background logging system using R's global calling handlers:

1. Verifies that the R version is 4.0.0 or higher, as global calling handlers are required.
2. Prompts the user to select a logging level: either "All Output" (Messages, Warnings, and Errors) or "Errors Only".
3. Prompts for a log filename, defaulting to a timestamped 'session_log_YYYYMMDD_HHMM.txt'.
4. Attaches 'globalCallingHandlers' to the session, which intercepts conditions and appends them to the log file with a timestamp and type label.

Value

Invisibly returns a named list with components: `status` ("done", "cancelled", or "error"), `log_file`, and `log_level`.

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  setup_sentinel()
}
```

`simulate_clean_room` *Clean-Room Simulator*

Description

Runs a specified R script in a background vanilla R session to verify reproducibility. If the script fails due to missing dependencies or variables, it interactively prompts the user to inject the necessary fixes directly into the file.

Usage

```
simulate_clean_room()
```

Details

The function implements a "stress-test" for script reproducibility:

1. Prompts the user to select an R script from the current directory.
2. Executes the script using ‘system2("Rscript", args = c("–vanilla", ...))‘, ensuring no workspace variables or attached packages from the current session interfere.
3. If the script crashes, it captures the ‘stderr‘ output and presents a crash report to the user.
4. Interactively offers to fix the crash by:
   - Injecting a missing ‘library()‘ call at the top of the file.
   - Injecting a custom code snippet (e.g., a missing variable definition).
5. Automatically updates the file and re-runs the simulation until the script executes successfully or the user aborts.

Value

Invisibly returns a named list with components: `status` ("done" or "cancelled"), `script`, `success` (logical), and `attempts` (integer).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  simulate_clean_room()
}
```

`sweep_memory` *Interactive Memory Sweeper*

Description

Scans the global environment for memory-intensive objects and interactively prompts the user to remove them to free up system RAM.

Usage

```
sweep_memory()
```

Details

The function implements a simple memory management workflow:

1. Prompts the user to define a size threshold (in MB) for flagging objects.
2. Calculates the size of all objects currently residing in the global environment.
3. Identifies and sorts objects that exceed the specified threshold.
4. Presents a selection menu allowing the user to choose one or more large objects for removal.
5. Executes ‘rm()‘ on the selected objects and immediately calls ‘gc()‘ to ensure the memory is released back to the system.

Value

Invisibly returns a named list with components: `status` ("done", "clean", or "cancelled"), `threshold_mb` (numeric), and `objects_removed` (character vector of removed object names).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  sweep_memory()
}
```

`sweep_temp_cache` *Temp-Cache Janitor*

Description

Scans hidden R temporary directories for abandoned session data and caches, categorizes the storage waste, and interactively prompts the user for safe deletion to reclaim disk space.

Usage

```
sweep_temp_cache()
```

Details

The function performs a deep scan of the OS-level temporary directory where R stores session data:

1. Identifies all directories matching the 'RtmpXXXXXX' pattern.
2. Scans these directories and categorizes files into buckets:
   - **Knitr/RMarkdown Caches**: '.knit.md', '.utf8.md', and '_cache' files.
   - **Downloaded Packages**: '.tar.gz', '.zip', and '.tgz' files.
   - **Raster/Image Files**: '.tif', '.png', '.jpg', and '.grd' files.
   - **General Session Data**: All other temporary files.
3. Calculates the total size of each bucket in megabytes.
4. Interactively prompts the user to select which buckets to permanently flush.

Value

Invisibly returns a named list with components: `status` ("done", "clean", or "cancelled"), `dirs_found` (integer), and `space_freed_mb` (numeric).

Warning

This function modifies files on disk or the global environment. Please ensure you have a backup or are using version control (e.g., Git) before execution.

Examples

```
if (interactive()) {
  sweep_temp_cache()
}
```

Index