Package 'chores'

December 9, 2025

Title A Collection of Large Language Model Assistants

Version 0.3.0

Description Provides a collection of ergonomic large language model assistants designed to help you complete repetitive, hard-to-automate tasks quickly. After selecting some code, press the keyboard shortcut you've chosen to trigger the package app, select an assistant, and watch your chore be carried out. While the package ships with a number of chore helpers for R package development, users can create custom helpers just by writing some instructions in a markdown file.

License MIT + file LICENSE

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/simonpcouch/chores, https://simonpcouch.github.io/chores/

BugReports https://github.com/simonpcouch/chores/issues

Imports cli (>= 3.6.3), glue (>= 1.8.0), ellmer (>= 0.4.0), miniUI (>= 0.1.1.1), rlang (>= 1.1.4), rstudioapi (>= 0.17.1), shiny (>= 1.9.1), streamy (>= 0.2.1)

Suggests gt, knitr, rmarkdown, testthat (>= 3.0.0), tibble, withr

VignetteBuilder knitr

NeedsCompilation no

Author Simon Couch [aut, cre] (ORCID: <https://orcid.org/0000-0001-5676-5107>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Simon Couch <simon.couch@posit.co>

Repository CRAN

Date/Publication 2025-12-09 21:30:01 UTC

Contents

`.init_addin` *Run the chores addin*

Description

The chores addin allows users to interactively select a chore helper to interface with the current selection. **This function is not intended to be interfaced with in regular usage of the package.** To launch the chores addin in RStudio, navigate to `Addins > Chores` and/or register the addin with a shortcut via `Tools > Modify Keyboard Shortcuts > Search "Chores"`–we suggest `Ctrl+Alt+C` (or `Ctrl+Cmd+C` on macOS).

Usage

```
.init_addin()
```

Value

`NULL`, invisibly. Called for the side effect of launching the chores addin and interfacing with selected text.

Examples

```
if (interactive()) {
  .init_addin()
}
```

`.init_helper` *Initialize a Helper object*

Description

Users typically should not need to call this function.

- Create new helpers that will automatically be registered with this function with `prompt_new()`.
- The chores addin will initialize needed helpers on-the-fly.

Usage

```
.init_helper(chore = NULL, .chores_chat = get_chores_chat())
```

Arguments

`chore`	The identifier for a helper prompt. By default one of "cli", "testthat" or "roxygen", though custom helpers can be added with `prompt_new()`.
`.chores_chat`	An ellmer Chat, e.g. `ellmer::chat_claude()`. Defaults to the `chores.chat` option, so e.g. set `options(chores.chat = ellmer::chat_claude(model = "claude-3-7-sonnet-20250219"))` in your `.Rprofile` to configure chores with ellmer every time you start a new R session.

Value

A Helper object, which is a subclass of an ellmer chat.

Examples

```
# requires an API key and sets options
## Not run:
# to create a chat with claude:
.init_helper(.chores_chat = ellmer::chat_claude(model = "claude-3-7-sonnet-20250219"))

# or with OpenAI's GPT-4o-mini:
.init_helper(.chores_chat = ellmer::chat_openai(model = "gpt-4o-mini"))

# to set OpenAI's GPT-4o-mini as the default model powering chores, for example,
# set the following option (possibly in your .Rprofile, if you'd like
# them to persist across sessions):
options(
  chores.chat = ellmer::chat_openai(model = "gpt-4o-mini")
)

## End(Not run)
```

`cli_helper` *The cli helper*

Description

A couple years ago, the tidyverse team began migrating to the cli R package for raising errors, transitioning away from base R (e.g. `stop()`), rlang (e.g. `rlang::abort()`), glue, and homegrown combinations of them. cli's new syntax is easier to work with as a developer and more visually pleasing as a user.

In some cases, transitioning is as simple as Finding + Replacing `rlang::abort()` to `cli::cli_abort()`. In others, there's a mess of ad-hoc pluralization, `paste0()`s, glue interpolations, and other assorted nonsense to sort through. Total pain, especially with thousands upon thousands of error messages thrown across the tidyverse, r-lib, and tidymodels organizations.

The cli helper helps you convert your R package to use cli for error messages.

Cost

The system prompt for a cli helper includes something like 2,400 tokens. Add in (a generous) 100 tokens for the code that's actually highlighted and also sent off to the model and you're looking at 2,500 input tokens. The model returns approximately the same number of output tokens as it receives, so we'll call that 100 output tokens per refactor.

As of the time of writing (October 2024), the recommended chores model Claude Sonnet 3.5 costs $3 per million input tokens and $15 per million output tokens. So, using the recommended model, **cli helpers cost around $9 for every 1,000 refactored pieces of code**. GPT-4o Mini, by contrast, doesn't tend to get cli markup classes right but *does* return syntactically valid calls to cli functions, and it would cost around 45 cents per 1,000 refactored pieces of code.

Gallery

This section includes a handful of examples "from the wild" and are generated with the recommended model, Claude Sonnet 3.5.

At its simplest, a one-line message with a little bit of markup:

```
rlang::abort("`save_pred` can only be used if the initial results saved predictions.")
```

Returns:

```
cli::cli_abort("{.arg save_pred} can only be used if the initial results saved predictions.")
```

Some strange vector collapsing and funky line breaking:

```
extra_grid_params <- glue::single_quote(extra_grid_params)
extra_grid_params <- glue::glue_collapse(extra_grid_params, sep = ", ")

msg <- glue::glue(
```

```
  "The provided `grid` has the following parameter columns that have ",
  "not been marked for tuning by `tune()`: {extra_grid_params}."
)

rlang::abort(msg)
```

Returns:

```
cli::cli_abort(
  "The provided {.arg grid} has parameter columns that have not been
   marked for tuning by {.fn tune}: {.val {extra_grid_params}}."
)
```

A message that probably best lives as two separate elements:

```
rlang::abort(
  paste(
    "Some model parameters require finalization but there are recipe",
    "parameters that require tuning. Please use ",
    "`extract_parameter_set_dials()` to set parameter ranges ",
    "manually and supply the output to the `param_info` argument."
  )
)
```

Returns:

```
cli::cli_abort(
  c(
    "Some model parameters require finalization but there are recipe
     parameters that require tuning.",
    "i" = "Please use {.fn extract_parameter_set_dials} to set parameter
           ranges manually and supply the output to the {.arg param_info}
           argument."
  )
)
```

Gnarly ad-hoc pluralization:

```
msg <- "Creating pre-processing data to finalize unknown parameter"
unk_names <- pset$id[unk]
if (length(unk_names) == 1) {
  msg <- paste0(msg, ": ", unk_names)
} else {
  msg <- paste0(msg, "s: ", paste0("'", unk_names, "'", collapse = ", "))
}
rlang::inform(msg)
```

Returns:

```
cli::cli_inform(
  "Creating pre-processing data to finalize unknown parameter{?s}: {.val {unk_names}}"
)
```

Some `paste0()` wonk:

```
rlang::abort(paste0(
  "The workflow has arguments to be tuned that are missing some ",
  "parameter objects: ",
  paste0("'", pset$id[!params], "'", collapse = ", ")
))
```

Returns:

```
cli::cli_abort(
  "The workflow has arguments to be tuned that are missing some
   parameter objects: {.val {pset$id[!params]}}"
)
```

The model is instructed to only return a call to a cli function, so erroring code that's run conditionally can get borked:

```
cls <- paste(cls, collapse = " or ")
if (!fine) {
  msg <- glue::glue("Argument '{deparse(cl$x)}' should be a {cls} or NULL")
  if (!is.null(where)) {
    msg <- glue::glue(msg, " in `{where}`")
  }
  rlang::abort(msg)
}
```

Returns:

```
cli::cli_abort(
  "Argument {.code {deparse(cl$x)}} should be {?a/an} {.cls {cls}} or {.code NULL}{?in {where}}."
)
```

Note that `?in where` is not valid cli markup.

Sprintf-style statements aren't an issue:

```
abort(sprintf("No such '%s' function: `%s()`.", package, name))
```

Returns:

```
cli::cli_abort("No such {.pkg {package}} function: {.fn {name}}.")
```

Interfacing manually with the cli helper

Chore helpers are typically interfaced with via the chores addin. To call the cli helper directly, use:

```
helper_cli <- .init_helper("cli")
```

Then, to submit a query, run:

```
helper_cli$chat({x})
```

`directory` *The prompt directory*

Description

The chores package's prompt directory is a directory of markdown files that is automatically registered with the chores package on package load. `directory_*()` functions allow users to interface with the directory, making new "chores" available:

- `directory_path()` returns the path to the prompt directory.
- `directory_set()` changes the path to the prompt directory (by setting the option `chores.dir`).
- `directory_list()` enumerates all of the different prompts that currently live in the directory (and provides clickable links to each).

Functions prefixed with `prompt*()` allow users to conveniently create, edit, and delete the prompts in chores' prompt directory.

Usage

```
directory_load(dir = directory_path())

directory_list()

directory_path()

directory_set(dir)
```

Arguments

`dir`	Path to a directory of markdown files–see `Details` for more.

Value

- `directory_path()` returns the path to the prompt directory (which is not created by default unless explicitly requested by the user).
- `directory_set()` return the path to the new prompt directory.
- `directory_list()` returns the file paths of all of the prompts that currently live in the directory (and provides clickable links to each).
- `directory_load()` returns `NULL` invisibly.

Format of the prompt directory

Prompts are markdown files with the name `chore-interface.md`, where interface is one of "replace", "prefix" or "suffix". An example directory might look like:

```
/
|-- .config/
|   |-- chores/
|       |-- proofread-replace.md
|       |-- summarize-prefix.md
```

In that case, chores will register two custom helpers when you call `library(chores)`. One of them is for the "proofread" chore and will replace the selected text with a proofread version (according to the instructions contained in the markdown file itself). The other is for the "summarize" chore and will prefix the selected text with a summarized version (again, according to the markdown file's instructions). Note:

- Files without a `.md` extension are ignored.
- Files with a `.md` extension must contain only one hyphen in their filename, and the text following the hyphen must be one of `replace`, `prefix`, or `suffix`.

To load custom prompts every time the package is loaded, place your prompts in `directory_path()`. To change the prompt directory without loading the package, just set the `chores.dir` option with `options(chores.dir = some_dir)`. To load a directory of files that's not the prompt directory, provide a `dir` argument to `directory_load()`.

See Also

The "Custom helpers" vignette, at `vignette("custom", package = "chores")`,for more on adding your own helper prompts, sharing them with others, and using prompts from others.

Examples

```
# choose a path for the prompt directory
tmp_dir <- withr::local_tempdir()
directory_set(tmp_dir)

# print out the current prompt directory
directory_path()

# list out prompts currently in the directory
directory_list()

# create a prompt in the prompt directory
prompt_new("boop", "replace")

# view updated list of prompts
directory_list()
```

helper_options *Options used by the chores package*

Description

The chores package makes use of two notable user-facing options:

- `chores.chat` determines the underlying LLM powering each helper. See the "Choosing a model" section of `vignette("chores", package = "chores")` for more information. The legacy option `.chores_chat` is also supported.
- `chores.dir` is the directory where helper prompts live. See the helper directory help-page for more information. The legacy option `.chores_dir` is also supported.

prompt *Working with helper prompts*

Description

The chores package provides a number of tools for working on system *prompts*. System prompts are what instruct helpers on how to behave and provide information to live in the models' "short-term memory."
`prompt_*()` functions allow users to conveniently create, edit, remove, the prompts in chores' "prompt directory."

- `prompt_new()` creates a new markdown file that will automatically create a helper with the specified chore, prompt, and interface on package load. Specify a `contents` argument to prefill with contents from a markdown file on your computer or the web.
- `prompt_edit()` and `prompt_remove()` open and delete, respectively, the file that defines the given chore's system prompt.

The prompts you create with these functions will be automatically loaded when you next trigger the helper addin.

Usage

```
prompt_new(chore, interface, contents = NULL)

prompt_remove(chore)

prompt_edit(chore)
```

Arguments

chore	A single string giving a descriptor of the helper's functionality. Cand only contain letters and numbers.
interface	One of "replace", "prefix", or "suffix", describing how the helper will interact with the selection. For example, the cli helper "replace"s the selection, while the roxygen helper "prefixes" the selected code with documentation.
contents	Optional. Path to a markdown file with contents that will "pre-fill" the file. Anything file ending in .md or .markdown that can be read with readLines() is fair game; this could be a local file, a "raw" URL to a GitHub Gist or file in a GitHub repository, etc.

Value

Each prompt_*() function returns the file path to the created, edited, or removed filepath, invisibly.

See Also

The directory help-page for more on working with prompts in batch using directory_*() functions, and vignette("custom", package = "chores") for more on sharing helper prompts and using prompts from others.

Examples

```
if (interactive()) {
# create a new helper for chore `"boop"` that replaces the selected text:
prompt_new("boop")

# after closing the file, reopen with:
prompt_edit("boop")

# remove the prompt (next time the package is loaded) with:
prompt_remove("boop")

# pull prompts from files on local drives or the web with
# `prompt_new(contents)`. for example, here is a GitHub Gist:
# paste0(
#  "https://gist.githubusercontent.com/simonpcouch/",
#  "daaa6c4155918d6f3efd6706d022e584/raw/ed1da68b3f38a25b58dd9fdc8b9c258d",
#  "58c9b4da/summarize-prefix.md"
# )
#
# press "Raw" and then supply that URL as `contents` (you don't actually
# have to use the paste0() to write out the URL--we're just keeping
# the characters per line under 80):
prompt_new(
  chore = "summarize",
  interface = "prefix",
  contents =
    paste0(
      "https://gist.githubusercontent.com/simonpcouch/",
```

```
      "daaa6c4155918d6f3efd6706d022e584/raw/ed1da68b3f38a25b58dd9fdc8b9c258d",
      "58c9b4da/summarize-prefix.md"
    )
)
}
```

`roxygen_helper` *The roxygen helper*

Description

The roxygen helper prefixes the selected function with a minimal roxygen2 documentation template. The helper is instructed to only generate a subset of a complete documentation entry, to be then completed by a developer:

- Stub `@param` descriptions based on defaults and inferred types
- Stub `@returns` entry that describes the return value as well as important errors and warnings users might encounter.

Cost

The system prompt from a roxygen helper includes something like 1,000 tokens. Add in 200 tokens for the code that's actually highlighted and also sent off to the model and you're looking at 1,200 input tokens. The model returns maybe 10 to 15 lines of relatively barebones royxgen documentation, so we'll call that 200 output tokens per refactor.

As of the time of writing (October 2024), the recommended chores model Claude Sonnet 3.5 costs $3 per million input tokens and $15 per million output tokens. So, using the recommended model, **roxygen helpers cost around $4 for every 1,000 generated roxygen documentation entries**. GPT-4o Mini, by contrast, doesn't tend to infer argument types correctly as often and often fails to line-break properly, but *does* usually return syntactically valid documentation entries, and it would cost around 20 cents per 1,000 generated roxygen documentation entries.

Gallery

This section includes a handful of examples "from the wild" and are generated with the recommended model, Claude Sonnet 3.5.

Documenting a function factory:

```
deferred_method_transform <- function(lambda_expr, transformer, eval_env) {
  transformer <- enexpr(transformer)
  force(eval_env)

  unique_id <- new_id()
  env_bind_lazy(
    generators,
    !!unique_id := inject((!!transformer)(!!lambda_expr)),
    eval.env = eval_env
```

```
  )

  inject(
    function(...) {
      (!!generators)[[!!unique_id]](self, private, ...)
    }
  )
}
```

Returns:

```
#' #' Transform a deferred method
#'
#' @description
#' A short description...
#'
#' @param lambda_expr A lambda expression to transform.
#' @param transformer A transformer function or expression.
#' @param eval_env The environment in which to evaluate the transformer.
#'
#' @returns
#' A function that, when called, will evaluate the transformed lambda expression.
#' The returned function accepts `...` arguments which are passed to the generated function.
#'
#' @export
```

A function that may raise a condition:

```
set_default <- function(value, default, arg = caller_arg(value)) {
  if (is.null(value)) {
    if (!is_testing() || is_snapshot()) {
      cli::cli_inform("Using {.field {arg}} = {.val {default}}.")
    }
    default
  } else {
    value
  }
}
```

Returns:

```
#' Set default value
#'
#' @description
#' A short description...
#'
#' @param value A value to check.
#' @param default The default value to use if `value` is NULL.
```

```
#' @param arg Optional. The name of the argument being set.
#'
#' @returns
#' Returns `value` if it's not NULL, otherwise returns `default`.
#' Informs the user when using the default value.
#'
#' @export
```

A function with some tricky indexing:

```
find_index <- function(left, e_right) {
 if (!is.list(e_right) || !has_name(e_right, "index") || !is.numeric(e_right$index)) {
    return(NA)
  }

  matches_idx <- map_lgl(left, function(e_left) e_left$index == e_right$index)
  if (sum(matches_idx) != 1) {
    return(NA)
  }
  which(matches_idx)[[1]]
}
```

Returns:

```
#' Find matching index
#'
#' @description
#' A short description...
#'
#' @param left A list of elements, each expected to have an 'index' field.
#' @param e_right A list with an 'index' field to search for in `left`.
#'
#' @returns
#' The numeric index in `left` where `e_right$index` matches, or NA if not found
#' or if inputs are invalid. Returns NA if multiple matches are found.
#'
#' @export
```

Interfacing manually with the roxygen helper

Chore helpers are typically interfaced with via the chores addin. To call the roxygen helper directly, use:

```
helper_roxygen <- .init_helper("roxygen")
```

Then, to submit a query, run:

```
helper_roxygen$chat({x})
```

`testthat_helper` *The testthat helper*

Description

testthat 3.0.0 was released in 2020, bringing with it numerous changes that were both huge quality of life improvements for package developers and also highly breaking changes.

While some of the task of converting legacy unit testing code to testthat 3e is quite is pretty straightforward, other components can be quite tedious. The testthat helper helps you transition your R package's unit tests to the third edition of testthat, namely via:

- Converting to snapshot tests
- Disentangling nested expectations
- Transitioning from deprecated functions like `expect_known_*()`

Cost

The system prompt from a testthat helper includes something like 1,000 tokens. Add in (a generous) 100 tokens for the code that's actually highlighted and also sent off to the model and you're looking at 1,100 input tokens. The model returns approximately the same number of output tokens as it receives, so we'll call that 100 output tokens per refactor.

As of the time of writing (October 2024), the recommended chores model Claude Sonnet 3.5 costs $3 per million input tokens and $15 per million output tokens. So, using the recommended model, **testthat helpers cost around $4 for every 1,000 refactored pieces of code**. GPT-4o Mini, by contrast, doesn't tend to get many pieces of formatting right and often fails to line-break properly, but *does* usually return syntactically valid calls to testthat functions, and it would cost around 20 cents per 1,000 refactored pieces of code.

Gallery

This section includes a handful of examples "from the wild" and are generated with the recommended model, Claude Sonnet 3.5.

Testthat helpers convert `expect_error()` (and `*_warning()` and `*_message()` and `*_condition()`) calls to use `expect_snapshot()` when there's a regular expression present:

```
expect_warning(
 check_ellipses("exponentiate", "tidy", "boop", exponentiate = TRUE, quick = FALSE),
 "\\`exponentiate\\` argument is not supported in the \\`tidy\\(\\)\\` method for \\`boop\\` objects"
)
```

Returns:

```
expect_snapshot(
  .res <- check_ellipses(
    "exponentiate", "tidy", "boop", exponentiate = TRUE, quick = FALSE
  )
)
```

Note, as well, that intermediate results are assigned to an object so as not to be snapshotted when their contents weren't previously tests.

Another example with multiple, redudant calls:

```
augment_error <- "augment is only supported for fixest models estimated with feols, feglm, or femlm"
expect_error(augment(res_fenegbin, df), augment_error)
expect_error(augment(res_feNmlm, df), augment_error)
expect_error(augment(res_fepois, df), augment_error)
```

Returns:

```
expect_snapshot(error = TRUE, augment(res_fenegbin, df))
expect_snapshot(error = TRUE, augment(res_feNmlm, df))
expect_snapshot(error = TRUE, augment(res_fepois, df))
```

They know about `regexp = NA`, which means "no error" (or warning, or message):

```
expect_error(
  p4_b <- check_parameters(w4, p4_a, data = mtcars),
  regex = NA
)
```

Returns:

```
expect_no_error(p4_b <- check_parameters(w4, p4_a, data = mtcars))
```

They also know not to adjust calls to those condition expectations when there's a `class` argument present (which usually means that one is testing a condition from another package, which should be able to change the wording of the message without consequence):

```
expect_error(tidy(pca, matrix = "u"), class = "pca_error")
```

Returns:

```
expect_error(tidy(pca, matrix = "u"), class = "pca_error")
```

When converting non-erroring code, testthat helpers will assign intermediate results so as not to snapshot both the result and the warning:

```
expect_warning(
  tidy(fit, robust = TRUE),
  '"robust" argument has been deprecated'
)
```

Returns:

```
expect_snapshot(
  .res <- tidy(fit, robust = TRUE)
)
```

Nested expectations can generally be disentangled without issue:

```
expect_equal(
  fit_resamples(decision_tree(cost_complexity = 1), bootstraps(mtcars)),
 expect_warning(tune_grid(decision_tree(cost_complexity = 1), bootstraps(mtcars)))
)
```

Returns:

```
expect_snapshot({
  fit_resamples_result <- fit_resamples(decision_tree(cost_complexity = 1),
                                        bootstraps(mtcars))
  tune_grid_result <- tune_grid(decision_tree(cost_complexity = 1),
                                bootstraps(mtcars))
})
expect_equal(fit_resamples_result, tune_grid_result)
```

There are also a few edits the helper knows to make to third-edition code. For example, it transitions `expect_snapshot_error()` and friends to use `expect_snapshot(error = TRUE)` so that the error context is snapshotted in addition to the message itself:

```
expect_snapshot_error(
  fit_best(knn_pca_res, parameters = tibble(neighbors = 2))
)
```

Returns:

```
expect_snapshot(
  error = TRUE,
  fit_best(knn_pca_res, parameters = tibble(neighbors = 2))
)
```

Interfacing manually with the testthat helper

Chore helpers are typically interfaced with via the chores addin. To call the testthat helper directly, use:

```
helper_testthat <- .init_helper("testthat")
```

Then, to submit a query, run:

```
helper_testthat$chat({x})
```

Index