Package: promoutils (via r-universe)

June 3, 2026

Title Utilities for Promoting rOpenSci

Version 0.6.1

Description Utility functions and workflows for promoting community events and community members on rOpenSci website and social media.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports cli, clipr, dplyr, gert (>= 2.1.5), gh, glue, googledrive (>= 2.1.1), httr2 (>= 1.0.0), jsonlite, lubridate, memoise (>= 2.0.1), monarch (>= 0.1.0), purrr, readr, rlang, stringr, tidyr, yaml

Suggests gitcreds, httptest2, httpuv, keyring (>= 1.4.1), matomoR (>= 0.0.1), pandoc, testthat (>= 3.0.0), usethis (>= 3.1.0), withr

Config/testthat/edition 3

Config/ropensci/maintainer staff

Depends R (>= 4.2.0)

URL https://docs.ropensci.org/promoutils, https://github.com/ropensci-org/promoutils

BugReports https://github.com/ropensci-org/promoutils/issues

Remotes ebbertd/matomoR, ropensci-org/monarch

Config/pak/sysreqs cmake git make libicu-dev libuv1-dev libssl-dev libx11-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2026-06-03 18:14:29 UTC

RemoteUrl https://github.com/ropensci-org/promoutils

RemoteRef main

RemoteSha 3d2b5091956c7c16cf7f02c5bbbeface0c2a01bc

Contents

`.gh_cache` *Create a cached version of the GH api calls*

Description

Create a cached version of the GH api calls

Usage

```
.gh_cache(..., .token = key("github"), .max_rate = NULL)
```

Arguments

`...` Arguments passed on to `gh::gh`

`endpoint` GitHub API endpoint. Must be one of the following forms:

- `METHOD path`, e.g. `GET /rate_limit`,
- `path`, e.g. `/rate_limit`,
- `METHOD url`, e.g. `GET https://api.github.com/rate_limit`,
- `url`, e.g. `https://api.github.com/rate_limit`.

If the method is not supplied, will use `.method`, which defaults to `"GET"`.

`per_page,.per_page` Number of items to return per page. If omitted, will be substituted by `max(.limit, 100)` if `.limit` is set, otherwise determined by the API (never greater than 100).

`.destfile` Path to write response to disk. If `NULL` (default), response will be processed and returned as an object. If path is given, response will be written to disk in the form sent. gh writes the response to a temporary file, and renames that file to `.destfile` after the request was successful. The name of the temporary file is created by adding a `-<random>.gh-tmp` suffix to it, where `<random>` is an ASCII string with random characters. gh removes the temporary file on error.

`.overwrite` If `.destfile` is provided, whether to overwrite an existing file. Defaults to `FALSE`. If an error happens the original file is kept.

`.api_url` Github API url (default: https://api.github.com). Used if `endpoint` just contains a path. Defaults to `GITHUB_API_URL` environment variable if set.

`.method` HTTP method to use if not explicitly supplied in the `endpoint`.

`.limit` Number of records to return. This can be used instead of manual pagination. By default it is `NULL`, which means that the defaults of the GitHub API are used. You can set it to a number to request more (or less) records, and also to `Inf` to request all records. Note, that if you request many records, then multiple GitHub API calls are used to get them, and this can take a potentially long time.

`.accept` The value of the `Accept` HTTP header. Defaults to `"application/vnd.github.v3+jso` . If `Accept` is given in `.send_headers`, then that will be used. This parameter can be used to provide a custom media type, in order to access a preview feature of the API.

`.send_headers` Named character vector of header field values (except `Authorization`, which is handled via `.token`). This can be used to override or augment the default `User-Agent` header: `"https://github.com/r-lib/gh"`.

`.progress` Whether to show a progress indicator for calls that need more than one HTTP request.

`.params` Additional list of parameters to append to `....` It is easier to use this than `...` if you have your parameters in a list already.

`.max_wait` Maximum number of seconds to wait if rate limited. Defaults to 10 minutes.

`.token` Authentication token. Defaults to `gh_token()`.

`.max_rate` Maximum request rate in requests per second. Set this to automatically throttle requests.

Details

```
memoise::memoise(gh::gh)
```

`by_platform` *Arrange by platform*

Description

Arranges data in long by platform (linkedin and mastodon).

Usage

```
by_platform(df)
```

Arguments

`df` Data frame. Formatted data including social media handles.

Value

Data frame arranged by platform on which to advertise.

Examples

```
u <- uc_fetch() |>
  uc_fmt("2025-01-01") |>
  uc_handles() |>
  by_platform()
```

---

cw_checkin *Create draft message for checking in with Cohosts*

Description

The week before, prepare the slides and coworking document, then use this draft text to invite the cohost(s) to review via Slack or Email.

Usage

```
cw_checkin(which = "next", names = NULL, print = FALSE)
```

Arguments

which	Character/Date. "next" to fetch details on the next coworking session, "last" to fetch details on the last scheduled (in future) coworking session, or a Date fetch details for a specific coworking session.
names	Character. Names of cohost if overriding those in the event listing.
print	Logical. Whether to simply print the text to console instead of copying to the clipboard.

---

cw_checkin_event *Create draft message for checking in with Cohosts about the Event review*

Description

When an event has been drafted use this draft text to invite the cohost(s) to review via Slack or Email.

Usage

```
cw_checkin_event(which = "next", names = NULL, print = FALSE)
```

Arguments

which	Character/Date. "next" to fetch details on the next coworking session, "last" to fetch details on the last scheduled (in future) coworking session, or a Date fetch details for a specific coworking session.
names	Character. Names of cohost if overriding those in the event listing.
print	Logical. Whether to print the copied text to the console.

Examples

```
cw_checkin_event("2026-04-07")
```

---

cw_details *Fetch details about coworking sessions*

Description

Fetch details about coworking sessions

Usage

```
cw_details(which = "next")
```

Arguments

which	Character/Date. "next" to fetch details on the next coworking session, "last" to fetch details on the last scheduled (in future) coworking session, or a Date fetch details for a specific coworking session.

Value

Data frame with coworking event details

Examples

```
cw_details()
# cw_details("2023-11") # Only works for events in the future
```

cw_docs_link *Get link to Coworking docs*

Description

Creates the coworking doc if it doesn't exist and returns a link either way. Optionally creates PR adds link to event page.

Usage

```
cw_docs_link(which = "next", open_sites = TRUE, add = FALSE)
```

Arguments

`which`	Character/Date. "next" to fetch details on the next coworking session, "last" to fetch details on the last scheduled (in future) coworking session, or a Date fetch details for a specific coworking session.
`open_sites`	Logical. Open websites with relevant details?
`add`	Logical. Whether to initialize a PR and add the link to the events file.

Value

Google Docs link

cw_event *Create a draft event for coworking*

Description

Creates a draft coworking event for the roweb3 website. All details pulled from the coworking todo list issue in `rosadmin/comms`. See `cw_issue()`.

Usage

```
cw_event(date, dry_run = FALSE)
```

Arguments

`date`	Character/Date. Date of the coworking event to create
`dry_run`	Logical. Whether to really create the event or just return the text.

`cw_issue` *Create Coworking To-Dos*

Description

Create a GitHub issue in a repository listing the Coworking ToDos. If no date or timezone, picks the next appropriate date (first Tuesday of the month following an existing coworking issue) and next appropriate timezone (cycling through America, Europe, and Australia) automatically.

Usage

```
cw_issue(
  date = NULL,
  tz = NULL,
  theme = "XXXX",
  cohost = "XXXX",
  repo = "rosadmin/comms",
  dry_run = FALSE
)
```

Arguments

`date`	Character. Date of next event (if NULL picks next first Tuesday).
`tz`	Character. Timezone of next event (if NULL picks next in order).
`theme`	Character. Name of theme, if unknown, uses XXXX placeholder
`cohost`	Character. Name of cohost, if unknown, uses XXXX placeholder
`repo`	Character. GitHub repository including owner (e.g., `rosadmin/comms`)
`dry_run`	Logical. Whether to really create the issue or just return the text.

Value

Nothing

Examples

```
cw_issue(dry_run = TRUE)
```

`cw_slack_hour` *Schedule 1-hour before messages on rOpenSci Slack*

Description

Will only work if running between the time that the 1-week message was posted and the start of the coworking.

Usage

```
cw_slack_hour(
  user = "UNRAUCMTK",
  test_run = FALSE,
  dry_run = FALSE,
  print = FALSE,
  call = rlang::caller_env()
)
```

Arguments

`user`	Character. Slack user id.
`test_run`	Logical. Whether to run a test of this message posting to the test channel.
`dry_run`	Logical. Whether to do a dry run, print the message only.
`print`	Logical. Whether to simply print the text to console instead of copying to the clipboard.
`call`	Environment. Parent environment for messaging.

Value

Nothing

Examples

```
cw_slack_hour(test_run = TRUE)
cw_slack_hour(dry_run = TRUE)
```

cw_slides_link *Open and fetch link to coworking slides*

Description

Open and fetch link to coworking slides

Usage

```
cw_slides_link(open_site = TRUE)
```

Arguments

open_site	Logical. Open slides in browser?

Value

Link to slides

cw_socials *Create a draft post for coworking*

Description

Creates draft posts for Mastodon and LinkedIn (by opening issues on rosadmin/scheduled_socials) and Slack (by printing the post text and schedule).

Usage

```
cw_socials(
  date,
  who_masto,
  who_slack,
  who_linkedin,
  who_main_masto = "@steffilazerte@fosstodon.org",
  who_main_slack = "<@UNRAUCMTK>",
  who_main_linkedin = "Steffi LaZerte",
  posters_tz = "America/Winnipeg",
  test_run = FALSE,
  dry_run = FALSE,
  print = TRUE,
  branch = NULL
)
```

Arguments

`date`	Character/Date. Date of the coworking event (local)
`who_masto`	Character. The full mastodon handle for the cohost (i.e. `XXXX@XXXX.com`)
`who_slack`	Character. The full API Slack id for the cohost (i.e. `<@UXXXXXXXX>`)
`who_linkedin`	Character. The full LinkedIn handle for the cohost (i.e. `@XXXX`)
`who_main_masto`	Character. The full mastodon handle for the rOpenSci staff organizer.
`who_main_slack`	Character. The Slack id for the rOpenSci staff organizer (i.e., `<@UXXXXXXXX>`. Defaults to Steffi's id.
`who_main_linkedin`	Character. The full LinkedIn handle for the rOpenSci staff organizer.
`posters_tz`	Character. Timezone of poster. Required for getting the time at which to post Slack messages as these are posted in the local timezone
`test_run`	Logical. Whether to do a test run (i.e. post to a test area).
`dry_run`	Logical. Whether to do a dry run (i.e. don't post).
`print`	Logical. Whether to simply print the text to console instead of copying to the clipboard.
`branch`	Character. Branch name if not on main.

Examples

```
cw_socials("2023-07-04",
           who_masto = "@cohost@mastodon.org",
           who_linkedin = "Cohost the Best",
           who_slack = "<UXXXXXXX>",
           dry_run = TRUE)

## Not run:
cw_socials("2023-07-04", who_masto = "@cohost@mastodon.org", who_slack = "<UXXXXXXX>")

## End(Not run)
```

`dict_helpwanted` *Dictionary for help-wanted*

Description

Creates a data frame dictionary of different versions of wording to use in help-wanted posts depending on the language of the post.

Usage

```
dict_helpwanted()
```

Value

Data frame.

Examples

```
dict_helpwanted()
```

---

dict_usecases *Dictionary for usecases*

Description

Creates a data frame dictionary of different versions of wording to use in usecase posts depending on the language of the post.

Usage

```
dict_usecases()
```

Value

Data frame.

Examples

```
dict_usecases()
```

---

gh_issue_close *Close a GH issue*

Description

Closes a GH issue by number, repository and owner.

Usage

```
gh_issue_close(n, owner, repo, dry_run = FALSE)
```

Arguments

n	Numeric. Issue number to close.
owner	Character. GitHub username or organization owner of the repository.
repo	Character. Repository in which to create issue.
dry_run	Logical. Whether to do a dry run (i.e. don't post).

Examples

```
## Not run:
  # Closes the first issue in the "myrop" repository of "myhandle" on GitHub
  gh_issue_close(1, "myhandle", "myrepo")

## End(Not run)
```

`gh_issue_fetch` *Fetch issues from a GH repository*

Description

Fetches issues from a GitHub repository and returns them as a list. Pass on to `gh_issue_fmt()` for formatting.

Usage

```
gh_issue_fetch(
  state = "open",
  labels = NULL,
  since = NULL,
  owner = "rosadmin",
  repo = "scheduled_socials",
  issue = NULL,
  verbose = FALSE
)
```

Arguments

`state`	Character. Which issues to fetch: "open", "closed", "all"
`labels`	Character vector. Fetch only issues with these labels. (Optional)
`since`	Character/Date/datetime. Fetch only issues since this date/time. (Optional)
`owner`	Character. Owner of the repository
`repo`	Character. Name of the repository (name of the package)
`issue`	Numeric. Specific Issue number to fetch.
`verbose`	Logical. Show extra informative messages.

Value

List of issues

Examples

```
i <- gh_issue_fetch()
i <- gh_issue_fetch(verbose = TRUE)
```

gh_issue_fmt *Format issues list from GH*

Description

Format issues list from GH

Usage

```
gh_issue_fmt(
  i,
  which = c("title", "number", "body", "labels", "url", "created", "updated",
    "gh_user_issue")
)
```

Arguments

i	List of issues from `gh_issue_fetch()`
which	Which fields to include

Value

Issues formated as a data frame

Examples

```
i <- gh_issue_fetch()
i <- gh_issue_fmt(i, which = "title")
```

gh_issue_post *Create a GH issues post*

Description

Opens an issue on Github with title, body and labels.

Usage

```
gh_issue_post(
  title,
  body,
  labels,
  owner,
  repo,
  avoid_dups = TRUE,
  dry_run = FALSE,
  open_browser = TRUE
)
```

Arguments

`title`	Character. Title of the issue.
`body`	Character. Body text of the issue.
`labels`	Character vector. Vector of labels to assign.
`owner`	Character. GitHub username or organization owner of the repository.
`repo`	Character. Repository in which to create issue.
`avoid_dups`	Logical. If TRUE (default), avoids posting duplicate issues.
`dry_run`	Logical. Whether to do a dry run (i.e. don't post).
`open_browser`	Logical. Whether to open the issue in the browser.

Examples

```
gh_issue_post(
  title = "My issue",
  body = "Body of the issue",
  labels = "help-wanted",
  owner = "ropensci",
  repo = "weathercan",
  dry_run = TRUE
)
```

`help_fetch` *Fetch help wanted issues*

Description

Fetch help wanted issues

Usage

```
help_fetch(min_date, json = "/repos/rosadmin/help-wanted/contents/issues.json")
```

Arguments

`min_date`	Character/Date. Earliest date a help-wanted label was added to an issue to be included.
`json`	Character. Location of the issues.json file to use.

Value

Data frame of help wanted issues

Examples

```
h <- help_fetch("2025-01-01")
```

help_handles *Get social media handles for helpwanted issues*

Description

Get social media handles for helpwanted issues

Usage

```
help_handles(help, force_masto = FALSE)
```

Arguments

help	Data frame of help wanted issues.
force_masto	Logical. Passed to monarch::add_handles(). Whether or not to force a re-check of mastodon handles (good if you think they've changed or they are 'none' and you want to check again).

Value

Data frame of help wanted issues with social median handles added

Examples

```
h <- help_fetch("2025-01-01") |>
  help_handles()
```

help_post *Create help-wanted* socials_post_issue() *command*

Description

Create help-wanted socials_post_issue() command

Usage

```
help_post(help, date_time = NULL, dry_run = FALSE, print = FALSE)
```

Arguments

help	Data frame. Formatted help-wanted issues including social media handles and arranged by platform upon which to post, output of by_platform().
date_time	Character/Date. When to post. Defaults to next Thursday if NULL.
dry_run	Logical. Whether to do a dry run (i.e. don't post).
print	Logical. Whether to simply print the text to console instead of copying to the clipboard.

Value

Copies commands to clipboard, optionally prints if (`print = TRUE`).

Examples

```
h <- help_fetch("2025-05-23") |>
  help_handles() |>
  by_platform()

help_post(h, print = TRUE)  # For non-interactive examples
```

---

help_wanted_json *Create help wanted JSON file*

Description

Function created for rosadmin/help-wanted. Fetches ropensci repos with issues labelled help-wanted. Saves output to issues.json for use by helpwanted workflows including https://ropensci.org/help-wanted.

Usage

```
help_wanted_json()
```

Value

Creates/updates `issues.json`

---

keys_check *Key status of credentials*

Description

Check to see if you have set up all the keys required by promoutils.

Usage

```
keys_check()
```

Value

Message informing of status

Examples

```
keys_check()
```

keys_set *Set keys required for promoutils API access*

Description

This function guides uses through the finding and setting tokens so that promoutils can find them.

Usage

```
keys_set(type = NULL)
```

Arguments

type	Character Vector. Keys/tokens to set ("slack", "matomo", "linkedin", "github").

Value

Nothing, side effect of setting token credentials in the keyring or via `gitcreds::gitcres_set()` for GitHub tokens.

li_auth *Authorize rOpenSci client with LinkedIn*

Description

This authorizes the rOpenSci client with **your** credentials (and you must be part of the rOpenSci organization as an admin). Make sure to take note of the 'refresh_token' as that is what you'll add to your .Renviron file for local work, or the GitHub secrets for the comms/scheduled_socials workflow.

Usage

```
li_auth()
```

Details

This function authorizes with a redirect url of "http://localhost:1444/", this *must* be the same as that listed in the LinkedIn Developer App, https://www.linkedin.com/developers/apps.

If you retrieve a new token, you will have to put it in the .Renviron and the re-start your R session to continue

This function authorizes with the scopes:

- w_member_social (default)
- w_organization_social (special request)
- r_organization_social (special request)
- r_organization_admin (special request)

Value

httr2 authorization

References

- Refresh tokens API: https://learn.microsoft.com/en-us/linkedin/shared/authentication/programmatic-refresh-tokens

Examples

```
## Not run:
# Only run if you need to update the scopes or get a new token (otherwise
# you'll have to replace all your tokens)
t <- li_auth()
t$refresh_token

## End(Not run)
```

---

li_posts_read *Get a list of recent posts by rOpenSci*

Description

Get a list of recent posts by rOpenSci

Usage

```
li_posts_read(author)
```

Arguments

author	Character. LinkedIn URN id (e.g., rOpenSci's is "urn:li:organization:77132573")

Value

list of posts

References

- https://learn.microsoft.com/en-us/linkedin/shared/api-guide/concepts/urns
- https://learn.microsoft.com/en-us/linkedin/marketing/integrations/community-management/shares/posts-api

Examples

```
li_posts_read(ro_urn)$elements[[1]]$commentary |> cat()
```

li_posts_write *Post to LinkedIn*

Description

Post to LinkedIn

Usage

```
li_posts_write(author, body, dry_run = FALSE)
```

Arguments

author	Character. URN. Either yours (see `li_urn_me()` or rOpenSci's "urn:li:organization:77132573")
body	Character. The body of the post as you would like it to appear.
dry_run	Logical. TRUE to show what would be sent to the server without actually sending it.

Value

A string of the URN for the post id.

Examples

```
# Dry-run
id <- li_posts_write(
  author = ro_urn, # Post on behalf of rOpenSci
  body = "Testing out the LinkedIn API via R and httr2!",
  dry_run = TRUE)

## Not run:
# Real post
id <- li_posts_write(
  author = ro_urn, # Post on behalf of rOpenSci
  body = "Testing out the LinkedIn API via R and httr2!")

## End(Not run)
```

li_urn_me *Fetch your personal URN number*

Description

This is required to post on LinkedIn to your personal account (for rOpenSci, use the organization urn, "urn:li:organization:77132573"

Usage

```
li_urn_me()
```

Value

A string with your URN in the format of "urn:li:person:XXXX"

Examples

```
## Not run:
li_urn_me()

## End(Not run)
```

`matomo_all` *Download all Matomo historic views*

Description

Downloads all views from a specific year range (2010-2024 by defautl). Views can be read with `matomo_read()`.

Usage

```
matomo_all(year_range = 2010:2024, host = "https://ropensci.matomo.cloud")
```

Arguments

`year_range`	Numeric vector. Years to download views for.
`host`	Character. URL for the Matomo host to collect views for. Defaults to that of rOpenSci.

Details

Requires a Renviron "MATOMO_TOKEN" key representing your token for Matomo for the host you'd like to fetch data for.

Value

Nothing. Writes views to disk.

matomo_blogposts *Formats and filter Matomo views to blogposts*

Description

Formats and filter Matomo views to blogposts

Usage

```
matomo_blogposts(views)
```

Arguments

views	Data frame from matomo_read().

Value

Data frame of blog post views

Examples

```
matomo_read() |>
  matomo_blogposts()
```

matomo_dir *Cache folder to store matomo views*

Description

Cache folder to store matomo views

Usage

```
matomo_dir()
```

Value

Character file path

Examples

```
matomo_dir()
```

matomo_read *Read Matomo views saved to disk*

Description

Reads a saved set of views.

Usage

```
matomo_read()
```

Value

Data frame of view data.

Examples

```
matomo_read()
```

matomo_update *Update Matomo views*

Description

Update a specific year of Matomo blog views. Defaults to current year. Views can be read with `matomo_read()`.

Usage

```
matomo_update(
  year = lubridate::year(Sys.Date()),
  host = "https://ropensci.matomo.cloud"
)
```

Arguments

`year`	Numeric. Year to update
`host`	Character. URL for the Matomo host to collect views for. Defaults to that of rOpenSci.

Details

Requires a .Renviron "MATOMO_TOKEN" key representing your token for Matomo for the host you'd like to fetch data for.

Value

Nothing. Updates views to disk.

`next_date` *Find the next date*

Description

Given a date and a day of the week, Given a date return the next month's first Tuesday

Usage

```
next_date(month, which = "Tues", n = 1, call = rlang::caller_env())
```

Arguments

`month`	Character/Date. The current month. Date returned is the next month.
`which`	Character/Numeric. Which week day to return. Either number or abbreviated English weekday.
`n`	Numeric. The nth week to return (i.e. the 1st Tuesday if `n = 1` and `which = "Tues"`).
`call`	Environment. Calling environment for appropriate messages if used within another function.

Value

A date

Examples

```
# Get the next first Tuesday
next_date("2023-11-01")
next_date("2023-11-30")

# Get the next 3rd Tuesday
next_date("2023-11-01", n = 3)

# Oops
## Not run:
next_date("2023-11-01", n = 5)

## End(Not run)
```

pkg_authors *Get package author names*

Description

Get package author names

Usage

```
pkg_authors(package, pkgs)
```

Arguments

package	Character. Package name
pkgs	Data frame. Packages returned by pkgs_ru().

Value

Character name of maintainer

Examples

```
pkg_authors("weathercan", pkgs_ru())
```

pkgs_ru *List of packages and details from R-Universe API*

Description

List of packages and details from R-Universe API

Usage

```
pkgs_ru(universe = "ropensci")
```

Arguments

universe	Character. Universe to collect details from.

Value

Data frame of package details

Examples

```
pkgs_ru()
```

post_time *Get the next post date/time*

Description

Finds the next date/time to post by day of the week and hour.

Usage

```
post_time(day, hour)
```

Arguments

day	Character. Day of the week (e.g., "Monday")
hour	Numeric. Hour at which to post (e.g., 8)

Value

Date time as character (without timezone)

Examples

```
post_time("Wednesday", 8)
```

prs_list *List PRs*

Description

List open PRs by url, title and number, optionally matching to a title or branch name (ref)

Usage

```
prs_list(match = NULL, owner = "ropensci", repo = "roweb3")
```

Arguments

match	Character. String to match in the title or branch name
owner	Character. GitHub owner of the repository (defaults to 'ropensci')
repo	Character. GitHub repository name (defaults to 'roweb3')

Value

Data frame with PR url, number, title, and branch name ('ref')

Examples

```
prs_list("coworking") # List all coworking related (open) PRs
```

replace_emoji *Replace emoji codes with unicode*

Description

Replaces emoji codes like :tada: with unicode like .

Usage

```
replace_emoji(x)
```

Arguments

x	Character. Text string within which to replace codes

Value

Text string with emoji unicodes

Examples

```
replace_emoji("hi :tada: testing \n\n\n Whow !   \n\n\n :smile:")
replace_emoji(":link:")
```

ro_urn *rOpenSci linkedin organization URN*

Description

rOpenSci linkedin organization URN

Usage

```
ro_urn
```

Format

An object of class `character` of length 1.

slack_channels *List channels and their ids*

Description

List channels and their ids

Usage

```
slack_channels(channel = NULL, types = c("public_channel", "private_channel"))
```

Arguments

channel	Character. Channel Name.
types	Character. Type of channels, one or both of "public_channel" or "private_channel"

Value

Data frame of channel names and ids, or if `channel` provided, a single channel id.

Examples

```
slack_channels()
slack_channels("general")
chn <- slack_channels(types = "private_channel")
```

slack_cleanup *Clean up old scheduled messages*

Description

Removes previously scheduled messages from `#admin-scheduled` if after the posting date.

Usage

```
slack_cleanup()
```

Value

Nothing

Examples

```
slack_posts_write("testing cleanup",
                  when = Sys.time() + lubridate::seconds(600),
                  tz = Sys.timezone())
slack_scheduled_list()

slack_cleanup()
```

---

slack_message_rm *Remove a Slack message*

Description

Use with caution!

Usage

```
slack_message_rm(msg = NULL, channel = NULL, channel_id = NULL, ts)
```

Arguments

msg	Data frame. Output of slack_messages containing messages to remove. Should contain columns "channel" and "ts". Note that only the most recent message will be removed.
channel	Character. Channel Name. Not required if channel_id supplied.
channel_id	Character. Channel id.
ts	Numeric. Timestamp to identify message to remove

Value

Nothing. Side effect of removing message from channel

---

slack_messages *Get the last 100 messages from a channel*

Description

Requires one of channel OR channel_id

Usage

```
slack_messages(channel = NULL, channel_id = NULL)
```

Arguments

channel	Character. Channel Name. Not required if channel_id supplied.
channel_id	Character. Channel id.

Value

Data frame. Messages and details

Examples

```
slack_messages(channel_id = "C026GCWKA") # General
slack_messages(channel = "General")
```

---

slack_posts_write *Write Slack message*

Description

Write a Slack message for posting now or later.

Usage

```
slack_posts_write(
  body,
  when = "now",
  tz = "America/Winnipeg",
  channel = "#testing-api",
  dry_run = FALSE
)
```

Arguments

body	Character. Text of message to post.
when	Character or Date/time. When to post message (timezone is ignored, see tz).
tz	Character. Timezone of when.
channel	Character. Channel to post message to.
dry_run	Logical. Test run?

Details

See https://docs.slack.dev/messaging/formatting-message-text#special-mentions https://docs.slack.dev/messagin message-text#mentioning-users

Value

Success message

References

- https://docs.slack.dev/messaging/sending-and-scheduling-messages
- https://docs.slack.dev/messaging/sending-and-scheduling-messages#scheduling

Examples

```
slack_posts_write("testing on Tuesday")
slack_posts_write("testing more and more", when = Sys.time() + 3600, tz = "Europe/Paris")
slack_posts_write(
  paste(
  "Join us for Social Coworking and office hours next week!",
  "",
  ":grey_exclamation: Theme: TESTING",
  ":hourglass_flowing_sand: When: TODAY!",
  ":cookie: Hosted by: USER! and cohost HOST",
  "",
  "You can use this time for...",
  "- General coworking", sep = "\n"), when = "now")

# Dry runs
slack_posts_write("testing on Tuesday", dry_run = TRUE)
slack_posts_write(
  "testing [this cool link](https://mycoolsite.com)",
  dry_run = TRUE
)
```

`slack_scheduled_list` *List currently scheduled messages*

Description

Returns the currently scheduled messages with added details regarding timezones, etc.

Usage

```
slack_scheduled_list()
```

Value

Data frame of currently scheduled messages

Examples

```
slack_scheduled_list()
```

`slack_scheduled_rm` *Delete a scheduled message*

Description

Removes a currently scheduled messages.

Usage

```
slack_scheduled_rm(msg = NULL, channel = NULL, id = NULL)
```

Arguments

`msg`	Data frame. Output of `slack_list_scheduled()` containing messages to remove. Should contain columns "channel" and "id"
`channel`	Character. If no msg, the Channel of the message to be deleted.
`id`	Character. If no msg, the ID of the message to be deleted.

Value

Nothing

Examples

```
# Schedule message
slack_posts_write("Testing delete msg", when = (Sys.Date() + lubridate::days(2)))

# Confirm scheduled
slack_scheduled_list()

# Remove message
slack_scheduled_list() |>
  dplyr::filter(text == "Testing delete msg") |>
  slack_scheduled_rm()

# Confirm that removed
slack_scheduled_list()

## Not run:
slack_scheduled_rm(channel = "#testing-api", id = "Q08U4S3J6QG")

## End(Not run)
```

slack_user *Fetch details on a specific user*

Description

Fetch details on a specific user

Usage

```
slack_user(name, users = NULL)
```

Arguments

name	Character. String to match real name to
users	Data frame. Data frame of users from `slack_users()`.

Value

Data frame

Examples

```
slack_user("Steffi")
```

slack_users *Fetch a list of Slack users*

Description

Fetch a list of Slack users

Usage

```
slack_users()
```

Value

Data frame of Slack users including names and ids

Examples

```
u <- slack_users()
```

`socials_post_issue` *Create a draft issue to post to Mastodon and LinkedIn*

Description

Formats the body and title of an issue and posts it on "rosadmin/scheduled_socials". The issue will bed opened in a browser for editing and confirmation. Note that issues will not be posted until the labels "draft" and "needs-review" have been removed.

Usage

```
socials_post_issue(
  time,
  tz = "America/Winnipeg",
  title,
  body,
  where = "mastodon",
  avoid_dups = TRUE,
  add_hash = TRUE,
  dry_run = FALSE,
  open_browser = TRUE,
  over_char_limit = cli::cli_abort,
  verbose = FALSE
)
```

Arguments

`time`	Date/time. Date and time at which the post should be made
`tz`	Character. Timezone (from `OlsonNames()`) in which to post
`title`	Character. Title of the post (`[Post]` and the date will be prepended and appended
`body`	Character. Text to be posted (omit the YAML for posting info; #RStats and @rstats@a.gup.pe will be appended for Mastodon, #RStats for LinkedIn), *or* link to text file with both Mastodon and LinkedIn body text, headed by by —- Mastodon —- and — LinkedIn —-.
`where`	Character vector. Either `mastodon` and/or `linkedin` to specify which platforms this should be posted on.
`avoid_dups`	Logical. Don't post an issue if any open issue has the same title.
`add_hash`	Logical. Whether to automatically add the RStats hashtags.
`dry_run`	Logical. Whether to perform a dry run (do not post, but display draft if `verbose = TRUE`).
`open_browser`	Logical. Whether to open the issue in the browser.
`over_char_limit`	Function. Error or warn if over the character limit?
`verbose`	Logical. Show extra informative messages.

Examples

```
socials_post_issue("2025-01-01 00:00:00",
                   title = "test post",
                   body = "Test body",
                   dry_run = TRUE, verbose = TRUE)
```

---

tt_post *Create Throwback Thursday* `socials_post_issue()` *command*

Description

Create Throwback Thursday `socials_post_issue()` command

Usage

```
tt_post(
  date,
  title,
  url,
  blurb = "<-- Interesting description/blurb mentioning the author -->",
  dry_run = FALSE,
  print = FALSE
)
```

Arguments

`date`	YMD Character. Date of the original post.
`title`	Character. Title of the original post.
`url`	Character. URL to original blog post
`blurb`	Character. Interesting bit to describe post (optional).
`dry_run`	Logical. Whether to do a dry run (i.e. don't post).
`print`	Logical. Whether to simply print the text to console instead of copying to the clipboard.

Value

Character string of R functions with relevant details (normally pasted into a POSTS_TT.R script file and tweaked as necessary before being executed).

Examples

```
# Standard
tt_post(
  "2019-05-14",
  "POWER to the People",
  "https://ropensci.org/blog/2019/05/14/nasapower/",
  print = TRUE
```

```
)

# Double throwback
tt_post(
  c("2017-08-22", "2017-08-22"),
  c("So you (don't) think you can review a package",
    "Onboarding visdat, a tool for preliminary visualisation of whole dataframes"
  ),
  url = c(
    "https://ropensci.org/blog/2017/08/22/first-package-review/",
    "https://ropensci.org/blog/2017/08/22/visdat/"
  ),
  print = TRUE
)

# Slug only
tt_post("2019-05-14", "POWER to the People", "nasapower", print = TRUE)
```

`tt_review` *Review blogposts views for Throwback Thursday highlights*

Description

Use `matomo_update()` and friends to download and update blog post view data. `tt_review()` summarizes the posts by view and optionally filters to a specific month.

Usage

```
tt_review(which_month = NULL)
```

Arguments

`which_month`	Numeric. Which month to return? Numeric 1-12 for month. Default `NULL` returns all.

Value

Data frame of posts by views with urls

Examples

```
tt_review(11)
```

uc_fetch *Fetch and format use cases from a GitHub Discussion board*

Description

Fetch and format use cases from a GitHub Discussion board

Usage

```
uc_fetch(owner = "ropensci", name = "discussions")
```

Arguments

owner	Character. Repository owner.
name	Character. Repository for discussions

Value

Data frame of usecases

Examples

```
u <- uc_fetch()
u
```

uc_fmt *Extract out resource information*

Description

Extract out resource information

Usage

```
uc_fmt(uc, min_date, pkgs = NULL)
```

Arguments

uc	Data frame. Data frame of usecases from `uc_fetch()`.
min_date	Character/Date. Minimum date of usecases to retain.
pkgs	Data frame. Optional data frame of package details (defaults to `pkgs_ru()`).

Value

Data frame of formatted use cases.

Examples

```
u <- uc_fetch() |>
  uc_fmt("2025-01-01")
u
```

`uc_handles` *Add social media handles*

Description

Add social media handles

Usage

```
uc_handles(uc, force_masto = FALSE)
```

Arguments

`uc`	Data frame. Formatted use cases output of `uc_fmt()`.
`force_masto`	Logical. Passed to `monarch::add_handles()`. Whether or not to force a re-check of mastodon handles (good if you think they've changed or they are 'none' and you want to check again).

Value

Data frame with social media handles appended/filled in

Examples

```
u <- uc_fetch() |>
  uc_fmt("2025-01-01") |>
  uc_handles()
```

uc_post *Create use case* socials_post_issue() *command*

Description

Create use case socials_post_issue() command

Usage

```
uc_post(uc, date_time = NULL, dry_run = FALSE, print = FALSE)
```

Arguments

uc	Data frame. Formatted use cases including social media handles and arranged by platform upon which to post, output of uc_platform().
date_time	Character/Date. When to post. Defaults to next Wednesday if NULL.
dry_run	Logical. Whether to do a dry run (i.e. don't post).
print	Logical. Whether to simply print the text to console instead of copying to the clipboard.

Value

Copies commands to clipboard, optionally prints if (print = TRUE).

Examples

```
u <- uc_fetch() |>
  uc_fmt("2025-01-01") |>
  uc_handles() |>
  by_platform()

uc_post(u, dry_run = TRUE,print = TRUE)
```

url_from_path *Create url from content date and slug*

Description

Create url from content date and slug

Usage

```
url_from_path(
  path,
  date = NULL,
  lang = NULL,
  where = "blog",
  base_url = "https://ropensci.org"
)
```

Arguments

path	Character. Slug, URL, or path to file in repository
date	Character. Date for event, used to create link (otherwise extracted from slug)
lang	Character. Language of blogpost (i.e., en, es, fr, etc.)
where	Character. 'blog' or 'event' depending on the content type.
base_url	Character. Base url of blog posts.

Value

Full url

Examples

```
url_from_path("my-post", date = "2025-01-01")
url_from_path("2025-01-01-my-post/index.es.md")
url_from_path("content/blog/2025-09-29-news-september-2025/index.md")
url_from_path("content/blog/2025-09-29-news-september-2025/index.Rmd")
url_from_path("content/blog/2025-09-29-news-september-2025/")
```

---

yaml_extract *Extract YAML keys from block*

Description

Extract YAML keys from block

Usage

```
yaml_extract(yaml, trim = "~~~")
```

Arguments

yaml	Character. String from which to extract YAML keys
trim	Character. Text to remove from the YAML block before processing. Usually the text that defines the block.

Value

data frame of yaml keys

Examples

```
yaml_extract("~~~start: 2023-11-12\nauthor: Steffi\n~~~")
```

Index