Package: googledrive (via r-universe)

October 29, 2026

Title An Interface to Google Drive

Version 2.2.0

Description Manage Google Drive files from R.

License MIT + file LICENSE

URL https://googledrive.tidyverse.org, https://github.com/tidyverse/googledrive

BugReports https://github.com/tidyverse/googledrive/issues

Depends R (>= 4.2)

Imports cli (>= 3.0.0), gargle (>= 1.7.0), glue (>= 1.4.2), httr, jsonlite, lifecycle, magrittr, pillar (>= 1.9.0), purrr (>= 1.0.1), rlang (>= 1.2.0), tibble (>= 2.0.0), utils, uuid, vctrs (>= 0.3.0), withr

Suggests curl, dplyr (>= 1.0.0), spelling, testthat (>= 3.1.5)

Config/Needs/website quarto, tidyverse, tidyverse/tidytemplate

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

NeedsCompilation no

Author Lucy D'Agostino McGowan [aut], Jennifer Bryan [aut, cre] (ORCID: <https://orcid.org/0000-0002-6983-2759>), Posit Software, PBC [cph, fnd]

Maintainer Jennifer Bryan <jenny@posit.co>

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

Date/Publication 2026-10-29 14:31:22 UTC

RemoteUrl https://github.com/cran/googledrive

RemoteRef HEAD

RemoteSha 66f202821ef7d14b5ad431fbc9acdd3030c6536a

Contents

as_dribble . . . 3
as_shared_drive . . . 4
dribble . . . 5
dribble-checks . . . 5
drive_about . . . 7
drive_auth . . . 8
drive_auth_configure . . . 11
drive_browse . . . 12
drive_cp . . . 13
drive_create . . . 15
drive_deauth . . . 17
drive_download . . . 18
drive_empty_trash . . . 19
drive_endpoints . . . 19
drive_examples . . . 20
drive_extension . . . 21
drive_fields . . . 22
drive_find . . . 23
drive_get . . . 26
drive_has_token . . . 28
drive_id . . . 29
drive_link . . . 30
drive_ls . . . 30
drive_mime_type . . . 31
drive_mkdir . . . 32
drive_mv . . . 34
drive_publish . . . 36
drive_put . . . 37
drive_read_string . . . 39
drive_rename . . . 40
drive_reveal . . . 41
drive_rm . . . 44
drive_scopes . . . 45
drive_share . . . 46
drive_token . . . 48
drive_token_write . . . 48
drive_trash . . . 50
drive_update . . . 51
drive_upload . . . 52
drive_user . . . 54
googledrive-configuration . . . 55
request_generate . . . 57
request_make . . . 58
shared_drive_create . . . 60
shared_drive_find . . . 61
shared_drive_get . . . 62

shared_drive_rm 63
shared_drive_update 64
shared_drives 65
shortcut_create 67
shortcut_resolve 68

Index **71**

`as_dribble` *Coerce to a* `dribble`

Description

Converts various representations of Google Drive files into a `dribble`, the object used by googledrive to hold Drive file metadata. Files can be specified via:

- File path. File name is an important special case.
- File id. Mark with `as_id()` to distinguish from file path.
- Data frame or `dribble`. Once you've successfully used googledrive to identify the files of interest, you'll have a `dribble`. Pass it into downstream functions.
- List representing Files resource objects. Mostly for internal use.

This is a generic function.

For maximum clarity, get your files into a `dribble` (or capture file id) as early as possible. When specifying via path, it's best to include the trailing slash when you're targeting a folder. If you want the folder `foo`, say `foo/`, not `foo`.

Some functions, such as `drive_cp()`, `drive_mkdir()`, `drive_mv()`, and `drive_upload()`, can accept the new file or folder name as the last part of `path`, when `name` is not given. But if you say `a/b/c` (no trailing slash) and a folder `a/b/c/` already exists, it's unclear what you want. A file named `c` in `a/b/` or a file with default name in `a/b/c/`? You get an error and must make your intent clear.

Usage

```
as_dribble(x, ...)
```

Arguments

`x`	A vector of Drive file paths, a vector of file ids marked with `as_id()`, a list of Files Resource objects, or a suitable data frame.
`...`	Other arguments passed down to methods. (Not used.)

Examples

```
# create some files for us to re-discover by name or filepath
alfa <- drive_create("alfa", type = "folder")
bravo <- drive_create("bravo", path = alfa)

# as_dribble() can work with file names or paths
as_dribble("alfa")
as_dribble("bravo")
as_dribble("alfa/bravo")
as_dribble(c("alfa", "alfa/bravo"))

# specify the file id (substitute a real file id of your own!)
# as_dribble(as_id("0B0Gh-SuuA2nTOGZVTXZTREgwZ2M"))

# Clean up
drive_find("alfa") |> drive_rm()
```

---

as_shared_drive *Coerce to shared drive*

Description

Converts various representations of a shared drive into a `dribble`, the object used by googledrive to hold Drive file metadata. Shared drives can be specified via

- Name
- Shared drive id, marked with `as_id()` to distinguish from name
- Data frame or `dribble` consisting solely of shared drives
- List representing Drives resource objects (mostly for internal use)

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives. Learn more about shared drives.

This is a generic function.

Usage

```
as_shared_drive(x, ...)
```

Arguments

x	A vector of shared drive names, a vector of shared drive ids marked with `as_id()`, a list of Drives resource objects, or a suitable data frame.
...	Other arguments passed down to methods. (Not used.)

Examples

```
## Not run:
# specify the name
as_shared_drive("abc")

# specify the id (substitute one of your own!)
as_shared_drive(as_id("0AOPK1X2jaNckUk9PVA"))

## End(Not run)
```

`dribble` *dribble object*

Description

googledrive stores the metadata for one or more Drive files or shared drives as a `dribble`. It is a "Drive tibble" with one row per file or shared drive and, at a minimum, these columns:

- `name`: a character column containing file or shared drive names
- `id`: a character column of file or shared drive ids
- `drive_resource`: a list-column, each element of which is either a Files resource or a Drives resource object. Note there is no guarantee that all documented fields are always present. We do check if the `kind` field is present and equal to one of `drive#file` or `drive#drive`.

The `dribble` format is handy because it exposes the file name, which is good for humans, but keeps it bundled with the file's unique id and other metadata, which are needed for API calls.

In general, the `dribble` class will be retained even after manipulation, as long as the required variables are present and of the correct type. This works best for manipulations via the dplyr and vctrs packages.

See Also

`as_dribble()`

`dribble-checks` *Check facts about a dribble*

Description

Sometimes you need to check things about a `dribble`‘ or about the files it represents, such as:

- Is it even a dribble?
- Size: Does the dribble hold exactly one file? At least one file? No file?
- File type: Is this file a folder?
- File ownership and access: Is it mine? Published? Shared?

Usage

```
is_dribble(d)

no_file(d)

single_file(d)

some_files(d)

confirm_dribble(d, arg = caller_arg(d), call = caller_env())

confirm_single_file(d, arg = caller_arg(d), call = caller_env())

confirm_some_files(d, arg = caller_arg(d), call = caller_env())

is_folder(d)

is_shortcut(d)

is_folder_shortcut(d)

is_native(d)

is_parental(d)

is_mine(d)

is_shared_drive(d)
```

Arguments

d	A dribble.
arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

Examples

```
## most of us have multiple files or folders on Google Drive
d <- drive_find()
is_dribble(d)
no_file(d)
single_file(d)
some_files(d)

# this will error
# confirm_single_file(d)
```

```
confirm_some_files(d)
is_folder(d)
is_mine(d)
```

`drive_about` *Get info on Drive capabilities*

Description

Gets information about the user, the user's Drive, and system capabilities. This function mostly exists to power `drive_user()`, which extracts the most useful information (the information on current user) and prints it nicely.

Usage

```
drive_about()
```

Value

A list representation of a Drive about resource

See Also

Wraps the `about.get` endpoint:

- https://developers.google.com/drive/api/v3/reference/about/get

Examples

```
drive_about()

# explore the export formats available for Drive files, by MIME type
about <- drive_about()
about[["exportFormats"]] |>
  purrr::map(unlist)
```

`drive_auth` *Authorize googledrive*

Description

Authorize googledrive to view and manage your Drive files. This function is a wrapper around `gargle::token_fetch()`.

By default, you are directed to a web browser, asked to sign in to your Google account, and to grant googledrive permission to operate on your behalf with Google Drive. By default, with your permission, these user credentials are cached in a folder below your home directory, from where they can be automatically refreshed, as necessary. Storage at the user level means the same token can be used across multiple projects and tokens are less likely to be synced to the cloud by accident.

Usage

```
drive_auth(
  email = gargle::gargle_oauth_email(),
  path = NULL,
  subject = NULL,
  scopes = "drive",
  cache = gargle::gargle_oauth_cache(),
  use_oob = gargle::gargle_oob_default(),
  token = NULL
)
```

Arguments

`email` Optional. If specified, `email` can take several different forms:

- `"jane@gmail.com"`, i.e. an actual email address. This allows the user to target a specific Google identity. If specified, this is used for token lookup, i.e. to determine if a suitable token is already available in the cache. If no such token is found, `email` is used to pre-select the targeted Google identity in the OAuth chooser. (Note, however, that the email associated with a token when it's cached is always determined from the token itself, never from this argument).
- `"*@example.com"`, i.e. a domain-only glob pattern. This can be helpful if you need code that "just works" for both `alice@example.com` and `bob@example.com`.
- `TRUE` means that you are approving email auto-discovery. If exactly one matching token is found in the cache, it will be used.
- `FALSE` or `NA` mean that you want to ignore the token cache and force a new OAuth dance in the browser.

Defaults to the option named `"gargle_oauth_email"`, retrieved by `gargle_oauth_email()` (unless a wrapper package implements different default behavior).

`path` JSON identifying the service account, in one of the forms supported for the `txt` argument of `jsonlite::fromJSON()` (typically, a file path or JSON string).

subject
: An optional subject claim. Specify this if you wish to use the service account represented by `path` to impersonate the `subject`, who is a normal user. Before this can work, an administrator must grant the service account domain-wide authority. Identify the user to impersonate via their email, e.g. `subject = "user@example.com"`. Note that gargle automatically adds the non-sensitive `"https://www.googleapis.com/auth/userinfo.email"` scope, so this scope must be enabled for the service account, along with any other `scopes` being requested.

scopes
: One or more API scopes. Each scope can be specified in full or, for Drive API-specific scopes, in an abbreviated form that is recognized by `drive_scopes()`:

  - "drive" = "https://www.googleapis.com/auth/drive" (the default)
  - "full" = "https://www.googleapis.com/auth/drive" (same as "drive")
  - "drive.readonly" = "https://www.googleapis.com/auth/drive.readonly"
  - "drive.file" = "https://www.googleapis.com/auth/drive.file"
  - "drive.appdata" = "https://www.googleapis.com/auth/drive.appdata"
  - "drive.metadata" = "https://www.googleapis.com/auth/drive.metadata"
  - "drive.metadata.readonly" = "https://www.googleapis.com/auth/drive.metadata.readonly"
  - "drive.photos.readonly" = "https://www.googleapis.com/auth/drive.photos.readonly"
  - "drive.scripts" = "https://www.googleapis.com/auth/drive.scripts

  See https://developers.google.com/drive/api/guides/api-specific-auth for details on the permissions for each scope.

cache
: Specifies the OAuth token cache. Defaults to the option named `"gargle_oauth_cache"`, retrieved via `gargle_oauth_cache()`.

use_oob
: Whether to use out-of-band authentication (or, perhaps, a variant implemented by gargle and known as "pseudo-OOB") when first acquiring the token. Defaults to the value returned by `gargle_oob_default()`. Note that (pseudo-)OOB auth only affects the initial OAuth dance. If we retrieve (and possibly refresh) a cached token, `use_oob` has no effect.

  If the OAuth client is provided implicitly by a wrapper package, its type probably defaults to the value returned by `gargle_oauth_client_type()`. You can take control of the client type by setting `options(gargle_oauth_client_type = "web")` or `options(gargle_oauth_client_type = "installed")`.

token
: A token with class Token2.0 or an object of httr's class `request`, i.e. a token that has been prepared with `httr::config()` and has a Token2.0 in the `auth_token` component.

Details

Most users, most of the time, do not need to call `drive_auth()` explicitly – it is triggered by the first action that requires authorization. Even when called, the default arguments often suffice.

However, when necessary, `drive_auth()` allows the user to explicitly:

- Declare which Google identity to use, via an `email` specification.
- Use a service account token or workload identity federation via `path`.
- Bring your own `token`.

- Customize scopes.
- Use a non-default cache folder or turn caching off.
- Explicitly request out-of-band (OOB) auth via use_oob.

If you are interacting with R within a browser (applies to RStudio Server, Posit Workbench, Posit Cloud, and Google Colaboratory), you need OOB auth or the pseudo-OOB variant. If this does not happen automatically, you can request it explicitly with use_oob = TRUE or, more persistently, by setting an option via options(gargle_oob_default = TRUE).

The choice between conventional OOB or pseudo-OOB auth is determined by the type of OAuth client. If the client is of the "installed" type, use_oob = TRUE results in conventional OOB auth. If the client is of the "web" type, use_oob = TRUE results in pseudo-OOB auth. Packages that provide a built-in OAuth client can usually detect which type of client to use. But if you need to set this explicitly, use the "gargle_oauth_client_type" option:

```
options(gargle_oauth_client_type = "web")       # pseudo-OOB
# or, alternatively
options(gargle_oauth_client_type = "installed") # conventional OOB
```

For details on the many ways to find a token, see gargle::token_fetch(). For deeper control over auth, use drive_auth_configure() to bring your own OAuth client or API key. To learn more about gargle options, see gargle::gargle_options.

See Also

Other auth functions: drive_auth_configure(), drive_deauth(), drive_scopes()

Examples

```
# load/refresh existing credentials, if available
# otherwise, go to browser for authentication and authorization
drive_auth()

# see user associated with current token
drive_user()

# force use of a token associated with a specific email
drive_auth(email = "jenny@example.com")
drive_user()

# force the OAuth web dance
drive_auth(email = NA)

# use a 'read only' scope, so it's impossible to edit or delete files
drive_auth(scopes = "drive.readonly")

# use a service account token
drive_auth(path = "foofy-83ee9e7c9c48.json")
```

`drive_auth_configure` *Edit and view auth configuration*

Description

These functions give more control over and visibility into the auth configuration than `drive_auth()` does. `drive_auth_configure()` lets the user specify their own:

- OAuth client, which is used when obtaining a user token.
- API key. If googledrive is de-authorized via `drive_deauth()`, all requests are sent with an API key in lieu of a token.

See the `vignette("get-api-credentials", package = "gargle")` for more. If the user does not configure these settings, internal defaults are used.

`drive_oauth_client()` and `drive_api_key()` retrieve the currently configured OAuth client and API key, respectively.

Usage

```
drive_auth_configure(client, path, api_key)

drive_api_key()

drive_oauth_client()
```

Arguments

`client`	A Google OAuth client, presumably constructed via `gargle::gargle_oauth_client_from_json()`. Note, however, that it is preferred to specify the client with JSON, using the `path` argument.
`path`	JSON downloaded from Google Cloud Console, containing a client id and secret, in one of the forms supported for the `txt` argument of `jsonlite::fromJSON()` (typically, a file path or JSON string).
`api_key`	API key.

Value

- `drive_auth_configure()`: An object of R6 class gargle::AuthState, invisibly.
- `drive_oauth_client()`: the current user-configured OAuth client.
- `drive_api_key()`: the current user-configured API key.

See Also

Other auth functions: `drive_auth()`, `drive_deauth()`, `drive_scopes()`

Examples

```
# see and store the current user-configured OAuth client (probably `NULL`)
(original_client <- drive_oauth_client())

# see and store the current user-configured API key (probably `NULL`)
(original_api_key <- drive_api_key())

# the preferred way to configure your own client is via a JSON file
# downloaded from Google Developers Console
# this example JSON is indicative, but fake
path_to_json <- system.file(
  "extdata", "client_secret_installed.googleusercontent.com.json",
  package = "gargle"
)
drive_auth_configure(path = path_to_json)

# this is also obviously a fake API key
drive_auth_configure(api_key = "the_key_I_got_for_a_google_API")

# confirm the changes
drive_oauth_client()
drive_api_key()

# restore original auth config
drive_auth_configure(client = original_client, api_key = original_api_key)
```

---

drive_browse *Visit Drive file in browser*

Description

Visits a file on Google Drive in your default browser.

Usage

```
drive_browse(file = .Last.value)
```

Arguments

file	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with as_id(), or a dribble.

Value

Character vector of file hyperlinks, from drive_link(), invisibly.

Examples

```
drive_find(n_max = 1) |> drive_browse()
```

drive_cp *Copy a Drive file*

Description

Copies an existing Drive file into a new file id.

Usage

```
drive_cp(file, path = NULL, name = NULL, ..., overwrite = NA)
```

Arguments

file	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with as_id(), or a dribble.
path	Specifies target destination for the new file on Google Drive. Can be an actual path (character), a file id marked with as_id(), or a dribble. If path is a shortcut to a folder, it is automatically resolved to its target folder. If path is given as a path (as opposed to a dribble or an id), it is best to explicitly indicate if it's a folder by including a trailing slash, since it cannot always be worked out from the context of the call. By default, the new file has the same parent folder as the source file.
name	Character, new file name if not specified as part of path. This will force path to be interpreted as a folder, even if it is character and lacks a trailing slash. Defaults to "Copy of FILE-NAME".
...	Named parameters to pass along to the Drive API. Has dynamic dots semantics. You can affect the metadata of the target file by specifying properties of the Files resource via .... Read the "Request body" section of the Drive API docs for the associated endpoint to learn about relevant parameters.
overwrite	Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in drive_get(). • NA (default): Just do the operation, even if it results in multiple files with the same filepath. • TRUE: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use drive_update() or drive_put(). • FALSE: Error if there is any pre-existing file at the filepath. Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class dribble, a tibble with one row per file.

See Also

Wraps the files.copy endpoint:

- https://developers.google.com/drive/api/v3/reference/files/copy

Examples

```
# Target one of the official example files
(src_file <- drive_example_remote("chicken.txt"))

# Make a "Copy of" copy in your My Drive
cp1 <- drive_cp(src_file)

# Make an explicitly named copy, in a different folder, and star it.
# The starring is an example of providing metadata via `...`.
# `starred` is not an actual argument to `drive_cp()`,
# it just gets passed through to the API.
folder <- drive_mkdir("drive-cp-folder")
cp2 <- drive_cp(
  src_file,
  path = folder,
  name = "chicken-cp.txt",
  starred = TRUE
)
drive_reveal(cp2, "starred")

# `overwrite = FALSE` errors if file already exists at target filepath
# THIS WILL ERROR!
# drive_cp(src_file, name = "Copy of chicken.txt", overwrite = FALSE)

# `overwrite = TRUE` moves an existing file to trash, then proceeds
cp3 <- drive_cp(src_file, name = "Copy of chicken.txt", overwrite = TRUE)

# Delete all of our copies and the new folder!
drive_rm(cp1, cp2, cp3, folder)

# Target an official example file that's a csv file
(csv_file <- drive_example_remote("chicken.csv"))

# copy AND AT THE SAME TIME convert it to a Google Sheet
chicken_sheet <- drive_cp(
  csv_file,
  name = "chicken-sheet-copy",
  mime_type = drive_mime_type("spreadsheet")
)
# is it really a Google Sheet?
drive_reveal(chicken_sheet, "mime_type")$mime_type
```

```
# go see the new Sheet in the browser
# drive_browse(chicken_sheet)

# Clean up
drive_rm(chicken_sheet)
```

`drive_create` *Create a new blank Drive file*

Description

Creates a new blank Drive file. Note there are better options for these special cases:

- Creating a folder? Use `drive_mkdir()`.
- Want to upload existing local content into a new Drive file? Use `drive_upload()`.

Usage

```
drive_create(name, path = NULL, type = NULL, ..., overwrite = NA)
```

Arguments

`name`	Name for the new file or, optionally, a path that specifies an existing parent folder, as well as the new file name.
`path`	Target destination for the new item, i.e. a folder or a shared drive. Can be given as an actual path (character), a file id or URL marked with `as_id()`, or a `dribble`. Defaults to your "My Drive" root folder. If `path` is a shortcut to a folder, it is automatically resolved to its target folder.
`type`	Character. Create a blank Google Doc, Sheet or Slides by setting `type` to `document`, `spreadsheet`, or `presentation`, respectively. All non-`NULL` values for `type` are pre-processed with `drive_mime_type()`.
`...`	Named parameters to pass along to the Drive API. Has dynamic dots semantics. You can affect the metadata of the target file by specifying properties of the Files resource via `...`. Read the "Request body" section of the Drive API docs for the associated endpoint to learn about relevant parameters.
`overwrite`	Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in `drive_get()`. • `NA` (default): Just do the operation, even if it results in multiple files with the same filepath. • `TRUE`: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use `drive_update()` or `drive_put()`.

- FALSE: Error if there is any pre-existing file at the filepath.

Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class dribble, a tibble with one row per file.

See Also

Wraps the files.create endpoint:

- https://developers.google.com/drive/api/v3/reference/files/create

Examples

```
# Create a blank Google Doc named 'WordStar' in
# your 'My Drive' root folder and star it
wordstar <- drive_create("WordStar", type = "document", starred = TRUE)

# is 'WordStar' really starred? YES
purrr::pluck(wordstar, "drive_resource", 1, "starred")

# Create a blank Google Slides presentation in
# the root folder, and set its description
execuvision <- drive_create(
  "ExecuVision",
  type = "presentation",
  description = "deeply nested bullet lists FTW"
)

# Did we really set the description? YES
purrr::pluck(execuvision, "drive_resource", 1, "description")

# check out the new presentation
drive_browse(execuvision)

# Create folder 'b4xl' in the root folder,
# then create an empty new Google Sheet in it
b4xl <- drive_mkdir("b4xl")
drive_create("VisiCalc", path = b4xl, type = "spreadsheet")

# Another way to create a Google Sheet in the folder 'b4xl'
drive_create("b4xl/SuperCalc", type = "spreadsheet")

# Yet another way to create a new file in a folder,
# this time specifying parent `path` as a character
drive_create("Lotus 1-2-3", path = "b4xl", type = "spreadsheet")

# Did we really create those Sheets in the intended folder? YES
drive_ls("b4xl")
```

```
# `overwrite = FALSE` errors if file already exists at target filepath
# THIS WILL ERROR!
drive_create("VisiCalc", path = b4xl, overwrite = FALSE)

# `overwrite = TRUE` moves an existing file to trash, then proceeds
drive_create("VisiCalc", path = b4xl, overwrite = TRUE)

# Clean up
drive_rm(wordstar, b4xl, execuvision)
```

`drive_deauth` *Suspend authorization*

Description

Put googledrive into a de-authorized state. Instead of sending a token, googledrive will send an API key. This can be used to access public resources for which no Google sign-in is required. This is handy for using googledrive in a non-interactive setting to make requests that do not require a token. It will prevent the attempt to obtain a token interactively in the browser. The user can configure their own API key via `drive_auth_configure()` and retrieve that key via `drive_api_key()`. In the absence of a user-configured key, a built-in default key is used.

Usage

```
drive_deauth()
```

See Also

Other auth functions: `drive_auth()`, `drive_auth_configure()`, `drive_scopes()`

Examples

```
drive_deauth()
drive_user()

# in a deauth'ed state, we can still get metadata on a world-readable file
public_file <- drive_example_remote("chicken.csv")
public_file
# we can still download it too
drive_download(public_file)
```

`drive_download` *Download a Drive file*

Description

This function downloads a file from Google Drive. Native Google file types, such as Google Docs, Google Sheets, and Google Slides, must be exported to a conventional local file type. This can be specified:

- explicitly via `type`
- implicitly via the file extension of `path`
- not at all, i.e. rely on the built-in default

To see what export file types are even possible, see the Drive API documentation or the result of `drive_about()$exportFormats`. The returned dribble includes a `local_path` column.

Usage

```
drive_download(file, path = NULL, type = NULL, overwrite = FALSE)
```

Arguments

`file`	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with `as_id()`, or a `dribble`.
`path`	Character. Path for output file. If absent, the default file name is the file's name on Google Drive and the default location is working directory, possibly with an added file extension.
`type`	Character. Only consulted if `file` is a native Google file. Specifies the desired type of the exported file. Will be processed via `drive_mime_type()`, so either a file extension like `"pdf"` or a full MIME type like `"application/pdf"` is acceptable.
`overwrite`	A logical scalar. If local `path` already exists, do you want to overwrite it?

Value

An object of class `dribble`, a tibble with one row per file.

See Also

Download files, in the Drive API documentation.

Examples

```
# Target one of the official example files
(src_file <- drive_example_remote("chicken_sheet"))

# Download Sheet as csv, explicit type
downloaded_file <- drive_download(src_file, type = "csv")

# See local path to new file
downloaded_file$local_path

# Download as csv, type implicit in file extension
drive_download(src_file, path = "my_csv_file.csv")

# Download with default name and type (xlsx)
drive_download(src_file)

# Clean up
unlink(c("chicken_sheet.csv", "chicken_sheet.xlsx", "my_csv_file.csv"))
```

---

drive_empty_trash *Empty Drive Trash*

Description

Caution, this will permanently delete files in your Drive trash.

Usage

```
drive_empty_trash()
```

---

drive_endpoints *List Drive endpoints*

Description

The googledrive package stores a named list of Drive API v3 endpoints (or "methods", using Google's vocabulary) internally and these functions expose this data.

- `drive_endpoint()` returns one endpoint, i.e. it uses `[[`.
- `drive_endpoints()` returns a list of endpoints, i.e. it uses `[`.

The names of this list (or the `id` sub-elements) are the nicknames that can be used to specify an endpoint in `request_generate()`. For each endpoint, we store its nickname or `id`, the associated HTTP verb, the `path`, and details about the parameters. This list is derived programmatically from the Drive API v3 Discovery Document (`https://www.googleapis.com/discovery/v1/apis/drive/v3/rest`) using the approach described in the Discovery Documents section of the gargle vignette Request helper functions.

Usage

```
drive_endpoints(i = NULL)

drive_endpoint(i)
```

Arguments

i	The name(s) or integer index(ices) of the endpoints to return. i is optional for drive_endpoints() and, if not given, the entire list is returned.

Value

One or more of the Drive API v3 endpoints that are used internally by googledrive.

Examples

```
str(head(drive_endpoints(), 3), max.level = 2)
drive_endpoint("drive.files.delete")
drive_endpoint(4)
```

---

drive_examples *Example files*

Description

googledrive makes a variety of example files – both local and remote – available for use in examples and reprexes. These functions help you access the example files. See vignette("example-files", package = "googledrive") for more.

Usage

```
drive_examples_local(matches)

drive_examples_remote(matches)

drive_example_local(matches)

drive_example_remote(matches)
```

Arguments

matches	A regular expression that matches the name of the desired example file(s). This argument is optional for the plural forms (drive_examples_local() and drive_examples_remote()) and, if provided, multiple matches are allowed. The single forms (drive_example_local() and drive_example_remote()) require this argument and require that there is exactly one match.

Value

- For `drive_example_local()` and `drive_examples_local()`, one or more local filepaths.
- For `drive_example_remote()` and `drive_examples_remote()`, a `dribble`.

Examples

```
drive_examples_local() |> basename()
drive_examples_local("chicken") |> basename()
drive_example_local("imdb")


drive_examples_remote()
drive_examples_remote("chicken")
drive_example_remote("chicken_doc")
```

`drive_extension` *Lookup extension from MIME type*

Description

This is a helper to determinine which extension should be used for a file. Two types of input are acceptable:

- MIME types accepted by Google Drive.
- File extensions, such as "pdf", "csv", etc. (these are simply passed through).

Usage

```
drive_extension(type = NULL)
```

Arguments

`type`	Character. MIME type or file extension.

Value

Character. File extension.

Examples

```
## get the extension for mime type image/jpeg
drive_extension("image/jpeg")

## it's vectorized
drive_extension(c("text/plain", "pdf", "image/gif"))
```

`drive_fields` *Request partial resources*

Description

You may be able to improve the performance of your API calls by requesting only the metadata that you actually need. This function is primarily for internal use and is currently focused on the Files resource. Note that high-level googledrive functions assume that the `name`, `id`, and `kind` fields are included, at a bare minimum. Assuming that `resource = "files"` (the default), input provided via `fields` is checked for validity against the known field names and the validated fields are returned. To see a tibble containing all possible fields and a short description of each, call `drive_fields(expose())`.

`prep_fields()` prepares fields for inclusion as query parameters.

Usage

```
drive_fields(fields = NULL, resource = "files")

prep_fields(fields, resource = "files")
```

Arguments

`fields`	Character vector of field names. If `resource = "files"`, they are checked for validity. Otherwise, they are passed through.
`resource`	Character, naming the API resource of interest. Currently, only the Files resource is anticipated.

Value

`drive_fields()`: Character vector of field names. `prep_fields()`: a string.

See Also

Improve performance, in the Drive API documentation.

Examples

```
# get a tibble of all fields for the Files resource + indicator of defaults
drive_fields(expose())

# invalid fields are removed and throw warning
drive_fields(c("name", "parents", "ownedByMe", "pancakes!"))

# prepare fields for query
prep_fields(c("name", "parents", "kind"))
```

drive_find *Find files on Google Drive*

Description

This is the closest googledrive function to what you can do at https://drive.google.com: by default, you just get a listing of your files. You can also search in various ways, e.g., filter by file type or ownership or work with shared drives. This is a very powerful function. Together with the more specific drive_get(), this is the main way to identify files to target for downstream work. If you know you want to search within a specific folder or shared drive, use drive_ls().

Usage

```
drive_find(
  pattern = NULL,
  trashed = FALSE,
  type = NULL,
  n_max = Inf,
  shared_drive = NULL,
  corpus = NULL,
  ...
)
```

Arguments

pattern	Character. If provided, only the items whose names match this regular expression are returned. This is implemented locally on the results returned by the API.
trashed	Logical. Whether to search files that are not in the trash (trashed = FALSE, the default), only files that are in the trash (trashed = TRUE), or to search regardless of trashed status (trashed = NA).
type	Character. If provided, only files of this type will be returned. Can be anything that drive_mime_type() knows how to handle. This is processed by googledrive and sent as a query parameter.
n_max	Integer. An upper bound on the number of items to return. This applies to the results requested from the API, which may be further filtered locally, via the pattern argument.
shared_drive	Anything that identifies one specific shared drive: its name, its id or URL marked with as_id(), or a dribble. The value provided to shared_drive is pre-processed with as_shared_drive(). Read more about shared drives.
corpus	Character, specifying which collections of items to search. Relevant to those who work with shared drives and/or Google Workspace domains. If specified, must be one of "user", "drive" (requires that shared_drive also be specified), "allDrives", or "domain". Read more about shared drives.
...	Other parameters to pass along in the request. The most likely candidate is q. See below and the API's Search for files and folders guide.

Value

An object of class `dribble`, a tibble with one row per file.

File type

The `type` argument is pre-processed with `drive_mime_type()`, so you can use a few shortcuts and file extensions, in addition to full-blown MIME types. googledrive forms a search clause to pass to `q`.

Search parameters

Do advanced search on file properties by providing search clauses to the `q` parameter that is passed to the API via `...`. Multiple `q` clauses or vector-valued `q` are combined via 'and'.

Trash

By default, `drive_find()` sets `trashed = FALSE` and does not include files in the trash. Literally, it adds `q = "trashed = false"` to the query. To search *only* the trash, set `trashed = TRUE`. To see files regardless of trash status, set `trashed = NA`, which adds `q = "(trashed = true or trashed = false)"` to the query.

Sort order

By default, `drive_find()` sends `orderBy = "recency desc"`, so the top files in your result have high "recency" (whatever that means). To suppress sending `orderBy` at all, do `drive_find(orderBy = NULL)`. The `orderBy` parameter accepts sort keys in addition to `recency`, which are documented in the `files.list` endpoint. googledrive translates a snake_case specification of `order_by` into the lowerCamel form, `orderBy`.

Shared drives and domains

If you work with shared drives and/or Google Workspace, you can apply your search query to collections of items beyond those associated with "My Drive". Use the `shared_drive` or `corpus` arguments to control this. Read more about shared drives.

See Also

Wraps the `files.list` endpoint:

- https://developers.google.com/drive/api/v3/reference/files/list

Helpful resource for forming your own queries:

- https://developers.google.com/drive/api/v3/search-files

Examples

```
## Not run:
# list "My Drive" w/o regard for folder hierarchy
drive_find()

# filter for folders, the easy way and the hard way
drive_find(type = "folder")
drive_find(q = "mimeType = 'application/vnd.google-apps.folder'")

# filter for Google Sheets, the easy way and the hard way
drive_find(type = "spreadsheet")
drive_find(q = "mimeType='application/vnd.google-apps.spreadsheet'")

# files whose names match a regex
# the local, general, sometimes-slow-to-execute version
drive_find(pattern = "ick")
# the server-side, executes-faster version
# NOTE: works only for a pattern at the beginning of file name
drive_find(q = "name contains 'chick'")

# search for files located directly in your root folder
drive_find(q = "'root' in parents")
# FYI: this is equivalent to
drive_ls("~/")

# control page size or cap the number of files returned
drive_find(pageSize = 50)
# all params passed through `...` can be camelCase or snake_case
drive_find(page_size = 50)
drive_find(n_max = 58)
drive_find(page_size = 5, n_max = 15)

# various ways to specify q search clauses
# multiple q's
drive_find(
  q = "name contains 'TEST'",
  q = "modifiedTime > '2020-07-21T12:00:00'"
)
# vector q
drive_find(q = c("starred = true", "visibility = 'anyoneWithLink'"))

# default `trashed = FALSE` excludes files in the trash
# `trashed = TRUE` consults ONLY file in the trash
drive_find(trashed = TRUE)
# `trashed = NA` disregards trash status completely
drive_find(trashed = NA)

# suppress the default sorting on recency
drive_find(order_by = NULL, n_max = 5)

# sort on various keys
drive_find(order_by = "modifiedByMeTime", n_max = 5)
```

```
# request descending order
drive_find(order_by = "quotaBytesUsed desc", n_max = 5)

## End(Not run)
```

`drive_get` *Get Drive files by path or id*

Description

Retrieves metadata for files specified via `path` or via file `id`. This function is quite straightforward if you specify files by `id`. But there are some important considerations when you specify your target files by `path`. See below for more. If the target files are specified via `path`, the returned `dribble` will include a `path` column.

Usage

```
drive_get(path = NULL, id = NULL, shared_drive = NULL, corpus = NULL)
```

Arguments

`path`	Character vector of path(s) to get. Use a trailing slash to indicate explicitly that a path is a folder, which can disambiguate if there is a file of the same name (yes this is possible on Drive!). If `path` appears to contain Drive URLs or is explicitly marked with `as_id()`, it is treated as if it was provided via the `id` argument.
`id`	Character vector of Drive file ids or URLs (it is first processed with `as_id()`). If both `path` and `id` are non-`NULL`, `id` is silently ignored.
`shared_drive`	Anything that identifies one specific shared drive: its name, its id or URL marked with `as_id()`, or a `dribble`. The value provided to `shared_drive` is pre-processed with `as_shared_drive()`. Read more about shared drives.
`corpus`	Character, specifying which collections of items to search. Relevant to those who work with shared drives and/or Google Workspace domains. If specified, must be one of `"user"`, `"drive"` (requires that `shared_drive` also be specified), `"allDrives"`, or `"domain"`. Read more about shared drives.

Value

An object of class `dribble`, a tibble with one row per file. If the target files were specified via `path`, there will be a `path` column.

Getting by `path`

Google Drive does NOT behave like your local file system! File and folder names need not be unique, even at a given level of the hierarchy. This means that a single path can describe multiple files (or 0 or exactly 1).

A single file can also be compatible with multiple paths, i.e. one path could be more specific than the other. A file located at `~/alfa/bravo` can be found as `bravo`, `alfa/bravo`, and `~/alfa/bravo`. If all 3 of those were included in the input `path`, they would be represented by a **single** row in the output.

It's best to think of `drive_get()` as a setwise operation when using file paths. Do not assume that the `i`-th input path corresponds to row `i` in the output (although it often does!). If there's not a 1-to-1 relationship between the input and output, this will be announced in a message.

`drive_get()` performs just enough path resolution to uniquely identify a file compatible with each input `path`, for all `paths` at once. If you absolutely want the full canonical path, run the output of `drive_get()` through `drive_reveal(d, "path")`‘.

Files that you don't own

If you want to get a file via `path` and it's not necessarily on your My Drive, you may need to specify the `shared_drive` or `corpus` arguments to search other collections of items. Read more about shared drives.

See Also

To add path information to any `dribble` that lacks it, use `drive_reveal(d, "path")`. To list the contents of a folder, use `drive_ls()`. For general searching, use `drive_find()`.

Wraps the `files.get` endpoint and, if you specify files by name or path, also calls `files.list`:

- https://developers.google.com/drive/api/v3/reference/files/get
- https://developers.google.com/drive/api/v3/reference/files/list

Examples

```
# get info about your "My Drive" root folder
drive_get("~/")
# the API reserves the file id "root" for your root folder
drive_get(id = "root")
drive_get(id = "root") |> drive_reveal("path")

# set up some files to get by path
alfalfa <- drive_mkdir("alfalfa")
broccoli <- drive_upload(
  drive_example_local("chicken.txt"),
  name = "broccoli", path = alfalfa
)
drive_get("broccoli")
drive_get("alfalfa/broccoli")
drive_get("~/alfalfa/broccoli")
drive_get(c("broccoli", "alfalfa/", "~/alfalfa/broccoli"))

# Clean up
drive_rm(alfalfa)

## Not run:
# The examples below are indicative of correct syntax.
```

```
# But note these will generally result in an error or a
# 0-row dribble, unless you replace the inputs with paths
# or file ids that exist in your Drive.

# multiple names
drive_get(c("abc", "def"))

# multiple names, one of which must be a folder
drive_get(c("abc", "def/"))

# query by file id(s)
drive_get(id = "abcdefgeh123456789")
drive_get(as_id("abcdefgeh123456789"))
drive_get(id = c("abcdefgh123456789", "jklmnopq123456789"))

# apply to a browser URL for, e.g., a Google Sheet
my_url <- "https://docs.google.com/spreadsheets/d/FILE_ID/edit#gid=SHEET_ID"
drive_get(my_url)
drive_get(as_id(my_url))
drive_get(id = my_url)

# access the shared drive named "foo"
# shared_drive params must be specified if getting by path
foo <- shared_drive_get("foo")
drive_get(c("this.jpg", "that-file"), shared_drive = foo)
# shared_drive params are not necessary if getting by id
drive_get(as_id("123456789"))

# search all shared drives and other files user has accessed
drive_get(c("this.jpg", "that-file"), corpus = "allDrives")

## End(Not run)
```

`drive_has_token` *Is there a token on hand?*

Description

Reports whether googledrive has stored a token, ready for use in downstream requests.

Usage

```
drive_has_token()
```

Value

Logical.

See Also

Other low-level API functions: `drive_token()`, `request_generate()`, `request_make()`

Examples

```
drive_has_token()
```

`drive_id` `drive_id` *class*

Description

`drive_id` is an S3 class to mark strings as Drive file ids, in order to distinguish them from Drive file names or paths. `as_id()` converts various inputs into an instance of `drive_id`.

`as_id()` is a generic function.

Usage

```
as_id(x, ...)
```

Arguments

`x`	A character vector of file or shared drive ids or URLs, a `dribble`, or a suitable data frame.
`...`	Other arguments passed down to methods. (Not used.)

Value

A character vector bearing the S3 class `drive_id`.

Examples

```
as_id("123abc")
as_id("https://docs.google.com/spreadsheets/d/qawsedrf16273849/edit#gid=12345")

x <- drive_find(n_max = 3)
as_id(x)
```

drive_link *Retrieve Drive file links*

Description

Returns the "webViewLink" for one or more files, which is the "link for opening the file in a relevant Google editor or viewer in a browser".

Usage

```
drive_link(file)
```

Arguments

file	Something that identifies the file(s) of interest on your Google Drive. Can be a character vector of names/paths, a character vector of file ids or URLs marked with as_id(), or a dribble.

Value

Character vector of file hyperlinks.

Examples

```
# get a few files into a dribble
three_files <- drive_find(n_max = 3)

# get their browser links
drive_link(three_files)
```

drive_ls *List contents of a folder or shared drive*

Description

List the contents of a folder or shared drive, recursively or not. This is a thin wrapper around drive_find(), that simply adds one constraint: the search is limited to direct or indirect children of path.

Usage

```
drive_ls(path = NULL, ..., recursive = FALSE)
```

Arguments

path	Specifies a single folder on Google Drive whose contents you want to list. Can be an actual path (character), a file id or URL marked with as_id(), or a dribble. If it is a shared drive or is a folder on a shared drive, it must be passed as a dribble. If path is a shortcut to a folder, it is automatically resolved to its target folder.
...	Any parameters that are valid for drive_find().
recursive	Logical, indicating if you want only direct children of path (recursive = FALSE, the default) or all children, including indirect (recursive = TRUE).

Value

An object of class dribble, a tibble with one row per file.

Examples

```
## Not run:
# get contents of the folder 'abc' (non-recursive)
drive_ls("abc")

# get contents of folder 'abc' whose names contain the letters 'def'
drive_ls(path = "abc", pattern = "def")

# get all Google spreadsheets in folder 'abc'
# whose names contain the letters 'def'
drive_ls(path = "abc", pattern = "def", type = "spreadsheet")

# get all the files below 'abc', recursively, that are starred
drive_ls(path = "abc", q = "starred = true", recursive = TRUE)

## End(Not run)
```

---

drive_mime_type *Lookup MIME type*

Description

This is a helper to determine which MIME type should be used for a file. Three types of input are acceptable:

- Native Google Drive file types. Important examples:
  - "document" for Google Docs
  - "folder" for folders
  - "presentation" for Google Slides
  - "spreadsheet" for Google Sheets
- File extensions, such as "pdf", "csv", etc.
- MIME types accepted by Google Drive (these are simply passed through).

Usage

```
drive_mime_type(type = NULL)
```

Arguments

type	Character. Google Drive file type, file extension, or MIME type. Pass the sentinel `expose()` if you want to get the full table used for validation and lookup, i.e. all MIME types known to be relevant to the Drive API.

Value

Character. MIME type.

Examples

```
## get the mime type for Google Spreadsheets
drive_mime_type("spreadsheet")

## get the mime type for jpegs
drive_mime_type("jpeg")

## it's vectorized
drive_mime_type(c("presentation", "pdf", "image/gif"))

## see the internal tibble of MIME types known to the Drive API
drive_mime_type(expose())
```

---

drive_mkdir *Create a Drive folder*

Description

Creates a new Drive folder. To update the metadata of an existing Drive file, including a folder, use `drive_update()`.

Usage

```
drive_mkdir(name, path = NULL, ..., overwrite = NA)
```

Arguments

name	Name for the new folder or, optionally, a path that specifies an existing parent folder, as well as the new name.
path	Target destination for the new folder, i.e. a folder or a shared drive. Can be given as an actual path (character), a file id or URL marked with `as_id()`, or a `dribble`. Defaults to your "My Drive" root folder. If `path` is a shortcut to a folder, it is automatically resolved to its target folder.

`...` Named parameters to pass along to the Drive API. Has dynamic dots semantics. You can affect the metadata of the target file by specifying properties of the Files resource via `...`. Read the "Request body" section of the Drive API docs for the associated endpoint to learn about relevant parameters.

`overwrite` Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in `drive_get()`.

- `NA` (default): Just do the operation, even if it results in multiple files with the same filepath.
- `TRUE`: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use `drive_update()` or `drive_put()`.
- `FALSE`: Error if there is any pre-existing file at the filepath.

Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class `dribble`, a tibble with one row per file.

See Also

Wraps the `files.create` endpoint:

- https://developers.google.com/drive/api/v3/reference/files/create

Examples

```
# Create folder named 'ghi', then another below named it 'jkl' and star it
ghi <- drive_mkdir("ghi")
jkl <- drive_mkdir("ghi/jkl", starred = TRUE)

# is 'jkl' really starred? YES
purrr::pluck(jkl, "drive_resource", 1, "starred")

# Another way to create folder 'mno' in folder 'ghi'
drive_mkdir("mno", path = "ghi")

# Yet another way to create a folder named 'pqr' in folder 'ghi',
# this time with parent folder stored in a dribble,
# and setting the new folder's description
pqr <- drive_mkdir("pqr", path = ghi, description = "I am a folder")

# Did we really set the description? YES
purrr::pluck(pqr, "drive_resource", 1, "description")
```

```
# `overwrite = FALSE` errors if something already exists at target filepath
# THIS WILL ERROR!
drive_create("name-squatter-mkdir", path = ghi)
drive_mkdir("name-squatter-mkdir", path = ghi, overwrite = FALSE)

# `overwrite = TRUE` moves the existing item to trash, then proceeds
drive_mkdir("name-squatter-mkdir", path = ghi, overwrite = TRUE)

# list everything inside 'ghi'
drive_ls("ghi")

# Clean up
drive_rm(ghi)
```

`drive_mv` *Move a Drive file*

Description

Move a Drive file to a different folder, give it a different name, or both.

Usage

```
drive_mv(file, path = NULL, name = NULL, overwrite = NA)
```

Arguments

`file`	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with `as_id()`, or a `dribble`.
`path`	Specifies target destination for the file on Google Drive. Can be an actual path (character), a file id marked with `as_id()`, or a `dribble`. If `path` is a shortcut to a folder, it is automatically resolved to its target folder. If `path` is given as a path (as opposed to a `dribble` or an id), it is best to explicitly indicate if it's a folder by including a trailing slash, since it cannot always be worked out from the context of the call. By default, the file stays in its current folder.
`name`	Character, new file name if not specified as part of `path`. This will force `path` to be interpreted as a folder, even if it is character and lacks a trailing slash. By default, the file keeps its current name.
`overwrite`	Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in `drive_get()`. • `NA` (default): Just do the operation, even if it results in multiple files with the same filepath.

- TRUE: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use `drive_update()` or `drive_put()`.
- FALSE: Error if there is any pre-existing file at the filepath.

Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class `dribble`, a tibble with one row per file.

See Also

Makes a metadata-only request to the `files.update` endpoint:

- https://developers.google.com/drive/api/v3/reference/files/update

Examples

```
# create a file to move
file <- drive_example_remote("chicken.txt") |>
  drive_cp("chicken-mv.txt")

# rename it, but leave in current folder (root folder, in this case)
file <- drive_mv(file, "chicken-mv-renamed.txt")

# create a folder to move the file into
folder <- drive_mkdir("mv-folder")

# move the file and rename it again,
# specify destination as a dribble
file <- drive_mv(file, path = folder, name = "chicken-mv-re-renamed.txt")

# verify renamed file is now in the folder
drive_ls(folder)

# move the file back to root folder
file <- drive_mv(file, "~/")

# move it again
# specify destination as path with trailing slash
# to ensure we get a move vs. renaming it to "mv-folder"
file <- drive_mv(file, "mv-folder/")

# `overwrite = FALSE` errors if something already exists at target filepath
# THIS WILL ERROR!
drive_create("name-squatter-mv", path = "~/")
drive_mv(file, path = "~/", name = "name-squatter-mv", overwrite = FALSE)
```

```
# `overwrite = TRUE` moves the existing item to trash, then proceeds
drive_mv(file, path = "~/", name = "name-squatter-mv", overwrite = TRUE)

# Clean up
drive_rm(file, folder)
```

`drive_publish` *Publish native Google files*

Description

Publish (or un-publish) native Google files to the web. Native Google files include Google Docs, Google Sheets, and Google Slides. The returned `dribble` will have extra columns, `published` and `revisions_resource`. Read more in `drive_reveal()`.

Usage

```
drive_publish(file, ...)

drive_unpublish(file, ...)
```

Arguments

`file`	Something that identifies the file(s) of interest on your Google Drive. Can be a character vector of names/paths, a character vector of file ids or URLs marked with `as_id()`, or a `dribble`.
`...`	Name-value pairs to add to the API request body (see API docs linked below for details). For `drive_publish()`, we include `publishAuto = TRUE` and `publishedOutsideDomain = TRUE`, if user does not specify other values.

Value

An object of class `dribble`, a tibble with one row per file. There will be extra columns, `published` and `revisions_resource`.

See Also

Wraps the `revisions.update` endpoint:

- https://developers.google.com/drive/api/v3/reference/revisions/update

Examples

```
# Create a file to publish
file <- drive_example_remote("chicken_sheet") |>
  drive_cp()

# Publish file
file <- drive_publish(file)
file$published

# Unpublish file
file <- drive_unpublish(file)
file$published

# Clean up
drive_rm(file)
```

`drive_put` *PUT new media into a Drive file*

Description

PUTs new media into a Drive file, in the HTTP sense:

- If the file already exists, we replace its content.
- If the file does not already exist, we create a new file.

This is a convenience wrapper around `drive_upload()` and `drive_update()`. In pseudo-code:

```
target_filepath <- <determined from `path`, `name`, and `media`>
hits <- <get all Drive files at target_filepath>
if (no hits) {
  drive_upload(media, path, name, type, ...)
} else if (exactly 1 hit) {
  drive_update(hit, media, ...)
} else {
  ERROR
}
```

Usage

```
drive_put(media, path = NULL, name = NULL, ..., type = NULL)
```

Arguments

`media`	Character, path to the local file to upload.

path — Specifies target destination for the new file on Google Drive. Can be an actual path (character), a file id marked with as_id(), or a dribble.

If path is a shortcut to a folder, it is automatically resolved to its target folder.

If path is given as a path (as opposed to a dribble or an id), it is best to explicitly indicate if it's a folder by including a trailing slash, since it cannot always be worked out from the context of the call. By default, the file is created in the current user's "My Drive" root folder.

name — Character, new file name if not specified as part of path. This will force path to be interpreted as a folder, even if it is character and lacks a trailing slash. Defaults to the file's local name.

... — Named parameters to pass along to the Drive API. Has dynamic dots semantics. You can affect the metadata of the target file by specifying properties of the Files resource via .... Read the "Request body" section of the Drive API docs for the associated endpoint to learn about relevant parameters.

type — Character. If type = NULL, a MIME type is automatically determined from the file extension, if possible. If the source file is of a suitable type, you can request conversion to Google Doc, Sheet or Slides by setting type to document, spreadsheet, or presentation, respectively. All non-NULL values for type are pre-processed with drive_mime_type().

Value

An object of class dribble, a tibble with one row per file.

Examples

```
# create a local file to work with
local_file <- tempfile("drive_put_", fileext = ".txt")
writeLines(c("beginning", "middle"), local_file)

# PUT to a novel filepath --> drive_put() delegates to drive_upload()
file <- drive_put(local_file)

# update the local file
cat("end", file = local_file, sep = "\n", append = TRUE)

# PUT again --> drive_put() delegates to drive_update()
file <- drive_put(local_file)

# create a second file at this filepath
file2 <- drive_create(basename(local_file))

# PUT again --> ERROR
drive_put(local_file)

# Clean up
drive_find("drive_put_.+[.]txt") |> drive_rm()
unlink(local_file)
```

`drive_read_string` *Read the content of a Drive file*

Description

These functions return the content of a Drive file as either a string or raw bytes. You will likely need to do additional work to parse the content into a useful R object.

`drive_download()` is the more generally useful function, but for certain file types, such as comma-separated values (MIME type `text/csv`), it can be handy to read data directly from Google Drive and avoid writing to disk.

Just as for `drive_download()`, native Google file types, such as Google Sheets or Docs, must be exported as a conventional MIME type. See the help for `drive_download()` for more.

Usage

```
drive_read_string(file, type = NULL, encoding = NULL)

drive_read_raw(file, type = NULL)
```

Arguments

`file`	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with `as_id()`, or a `dribble`.
`type`	Character. Only consulted if `file` is a native Google file. Specifies the desired type of the exported file. Will be processed via `drive_mime_type()`, so either a file extension like `"pdf"` or a full MIME type like `"application/pdf"` is acceptable.
`encoding`	Passed along to `httr::content()`. Describes the encoding of the *input* `file`.

Value

- `read_drive_string()`: a UTF-8 encoded string
- `read_drive_raw()`: a `raw()` vector

Examples

```
# comma-separated values --> data.frame or tibble
(chicken_csv <- drive_example_remote("chicken.csv"))
read.csv(text = chicken_csv |> drive_read_string())

# Google Doc --> character vector
(chicken_doc <- drive_example_remote("chicken_doc"))
chicken_doc |>
  # NOTE: we must specify an export MIME type
  drive_read_string(type = "text/plain") |>
  strsplit(split = "(\r\n|\r|\n)")
  (\(x) x[[1]])()
```

drive_rename *Rename a Drive file*

Description

This is a wrapper for `drive_mv()` that only renames a file. If you would like to rename AND move the file, see `drive_mv()`.

Usage

```
drive_rename(file, name = NULL, overwrite = NA)
```

Arguments

`file`	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with `as_id()`, or a `dribble`.
`name`	Character. Name you would like the file to have.
`overwrite`	Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in `drive_get()`. • `NA` (default): Just do the operation, even if it results in multiple files with the same filepath. • `TRUE`: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use `drive_update()` or `drive_put()`. • `FALSE`: Error if there is any pre-existing file at the filepath. Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class `dribble`, a tibble with one row per file.

Examples

```
# Create a file to rename
file <- drive_create("file-to-rename")

# Rename it
file <- drive_rename(file, name = "renamed-file")

# `overwrite = FALSE` errors if something already exists at target filepath
```

```
# THIS WILL ERROR!
drive_create("name-squatter-rename")
drive_rename(file, name = "name-squatter-rename", overwrite = FALSE)

# `overwrite = TRUE` moves the existing item to trash, then proceeds
file <- drive_rename(file, name = "name-squatter-rename", overwrite = TRUE)

# Clean up
drive_rm(file)
```

`drive_reveal` *Add a new column of Drive file information*

Description

`drive_reveal()` adds extra information about your Drive files that is not readily available in the default `dribble` produced by googledrive. Why is this info not always included in the default `dribble`?

- You don't always care about it. There is a lot of esoteric information in the `drive_resource` that has little value for most users.
- It might be "expensive" to get this information and put it into a usable form. For example, revealing a file's `"path"`, `"permissions"`, or `"published"` status all require additional API calls.

`drive_reveal()` can also **hoist** any property out of the `drive_resource` list-column, when the property's name is passed as the `what` argument. The resulting new column is simplified if it is easy to do so, e.g., if the individual elements are all string or logical. If `what` extracts a date-time, we return `POSIXct`. Otherwise, you'll get a list-column. If this makes you sad, consider using `tidyr::hoist()` instead. It is more powerful due to a richer "plucking specification" and its `ptype` and `transform` arguments. Another useful function is `tidyr::unnest_wider()`.

Usage

```
drive_reveal(file, what = c("path", "permissions", "published", "parent"))
```

Arguments

`file`	Something that identifies the file(s) of interest on your Google Drive. Can be a character vector of names/paths, a character vector of file ids or URLs marked with `as_id()`, or a `dribble`.
`what`	Character, describing the type of info you want to add. These values get special handling (more details below): • `path` • `permissions` • `published`

- `parent`

You can also request any property in the `drive_resource` column by name. The request can be in `camelCase` or `snake_case`, but the new column name will always be `snake_case`. Some examples of `what`:

- `mime_type` (or `mimeType`)
- `trashed`
- `starred`
- `description`
- `version`
- `web_view_link` (or `webViewLink`)
- `modified_time` (or `modifiedTime`)
- `created_time` (or `createdTime`)
- `owned_by_me` (or `ownedByMe`)
- `size`
- `quota_bytes_used` (or `quotaBytesUsed`)

Value

An object of class `dribble`, a tibble with one row per file. The additional info requested via `what` appears in one (or more) extra columns.

File path

When `what = "path"` the `dribble` gains a character column holding each file's path. This can be *very slow*, so use with caution.

The example path `~/a/b/` illustrates two conventions used in googledrive:

- The leading `~/` means that the folder `a` is located in the current user's "My Drive" root folder.
- The trailing `/` means that `b`, located in `a`, is *a folder or a folder shortcut*.

Permissions

When `what = "permissions"` the `dribble` gains a logical column `shared` that indicates whether a file is shared and a new list-column `permissions_resource` containing lists of Permissions resources.

Publishing

When `what = "published"` the `dribble` gains a logical column `published` that indicates whether a file is published and a new list-column `revision_resource` containing lists of Revisions resources.

Parent

When `what = "parent"` the `dribble` gains a character column `id_parent` that is the file id of this item's parent folder. This information is available in the `drive_resource`, but can't just be hoisted out:

- Google Drive used to allow files to have multiple parents, but this is no longer supported and googledrive now assumes this is impossible. However, we have seen (very old) files that still have >1 parent folder. If we see this we message about it and drop all but the first parent.
- The `parents` property in `drive_resource` has an "extra" layer of nesting and needs to be flattened.

If you really want the raw `parents` property, call `drive_reveal(what = "parents")`.

See Also

To learn more about the properties present in the metadata of a Drive file (which is what's in the `drive_resource` list-column of a `dribble`), see the API docs:

- https://developers.google.com/drive/api/v3/reference/files#resource-representations

Examples

```
# Get a few of your files
files <- drive_find(n_max = 10, trashed = NA)

# the "special" cases that require additional API calls and can be slow
drive_reveal(files, "path")
drive_reveal(files, "permissions")
drive_reveal(files, "published")

# a "special" case of digging info out of `drive_resource`, then processing
# a bit
drive_reveal(files, "parent")

# the "simple" cases of digging info out of `drive_resource`
drive_reveal(files, "trashed")
drive_reveal(files, "mime_type")
drive_reveal(files, "starred")
drive_reveal(files, "description")
drive_reveal(files, "version")
drive_reveal(files, "web_view_link")
drive_reveal(files, "modified_time")
drive_reveal(files, "created_time")
drive_reveal(files, "owned_by_me")
drive_reveal(files, "size")
drive_reveal(files, "quota_bytes_used")

# 'root' is a special file id that represents your My Drive root folder
drive_get(id = "root") |>
  drive_reveal("path")
```

drive_rm *Delete files from Drive*

Description

Caution: this will permanently delete your files! For a safer, reversible option, see `drive_trash()`.

Usage

```
drive_rm(...)
```

Arguments

`...`	One or more Drive files, specified in any valid way, i.e. as a `dribble`, by name or path, or by file id or URL marked with `as_id()`. Or any combination thereof. Elements are processed with `as_dribble()` and row-bound prior to deletion.

Value

Logical vector, indicating whether the delete succeeded.

See Also

Wraps the `files.delete` endpoint:

- https://developers.google.com/drive/api/v3/reference/files/delete

Examples

```
# Target one of the official example files to copy (then remove)
(src_file <- drive_example_remote("chicken.txt"))

# Create a copy, then remove it by name
src_file |>
  drive_cp(name = "chicken-rm.txt")
drive_rm("chicken-rm.txt")

# Create several more copies
x1 <- src_file |>
  drive_cp(name = "chicken-abc.txt")
drive_cp(src_file, name = "chicken-def.txt")
x2 <- src_file |>
  drive_cp(name = "chicken-ghi.txt")

# Remove the copies all at once, specified in different ways
drive_rm(x1, "chicken-def.txt", as_id(x2))
```

drive_scopes *Produce scopes specific to the Drive API*

Description

When called with no arguments, `drive_scopes()` returns a named character vector of scopes associated with the Drive API. If `drive_scopes(scopes =)` is given, an abbreviated entry such as `"drive.readonly"` is expanded to a full scope (`"https://www.googleapis.com/auth/drive.readonly"` in this case). Unrecognized scopes are passed through unchanged.

Usage

```
drive_scopes(scopes = NULL)
```

Arguments

scopes	One or more API scopes. Each scope can be specified in full or, for Drive API-specific scopes, in an abbreviated form that is recognized by `drive_scopes()`: • "drive" = "https://www.googleapis.com/auth/drive" (the default) • "full" = "https://www.googleapis.com/auth/drive" (same as "drive") • "drive.readonly" = "https://www.googleapis.com/auth/drive.readonly" • "drive.file" = "https://www.googleapis.com/auth/drive.file" • "drive.appdata" = "https://www.googleapis.com/auth/drive.appdata" • "drive.metadata" = "https://www.googleapis.com/auth/drive.metadata" • "drive.metadata.readonly" = "https://www.googleapis.com/auth/drive.metadata.readonly" • "drive.photos.readonly" = "https://www.googleapis.com/auth/drive.photos.readonly" • "drive.scripts" = "https://www.googleapis.com/auth/drive.scripts See https://developers.google.com/drive/api/guides/api-specific-auth for details on the permissions for each scope.

Value

A character vector of scopes.

See Also

https://developers.google.com/drive/api/guides/api-specific-auth for details on the permissions for each scope.

Other auth functions: `drive_auth()`, `drive_auth_configure()`, `drive_deauth()`

Examples

```
drive_scopes("full")
drive_scopes("drive.readonly")
drive_scopes()
```

drive_share *Share Drive files*

Description

Grant individuals or other groups access to files, including permission to read, comment, or edit. The returned dribble will have extra columns, shared and permissions_resource. Read more in drive_reveal().

drive_share_anyone() is a convenience wrapper for a common special case: "make this file readable by 'anyone with a link'".

Usage

```
drive_share(
  file,
 role = c("reader", "commenter", "writer", "fileOrganizer", "owner", "organizer"),
  type = c("user", "group", "domain", "anyone"),
  ...
)

drive_share_anyone(file)
```

Arguments

file	Something that identifies the file(s) of interest on your Google Drive. Can be a character vector of names/paths, a character vector of file ids or URLs marked with as_id(), or a dribble.
role	Character. The role to grant. Must be one of: • owner (not allowed in shared drives) • organizer (applies to shared drives) • fileOrganizer (applies to shared drives) • writer • commenter • reader
type	Character. Describes the grantee. Must be one of: • user • group • domain • anyone
...	Name-value pairs to add to the API request. This is where you provide additional information, such as the emailAddress (when grantee type is "group" or "user") or the domain (when grantee type is "domain"). Read the API docs linked below for more details.

Value

An object of class dribble, a tibble with one row per file. There will be extra columns, shared and permissions_resource.

See Also

Wraps the permissions.create endpoint:

- https://developers.google.com/drive/api/v3/reference/permissions/create

Drive roles and permissions are described here:

- https://developers.google.com/drive/api/v3/ref-roles

Examples

```
# Create a file to share
file <- drive_example_remote("chicken_doc") |>
  drive_cp(name = "chicken-share.txt")

# Let a specific person comment
file <- file |>
  drive_share(
    role = "commenter",
    type = "user",
    emailAddress = "susan@example.com"
  )

# Let a different specific person edit and customize the email notification
file <- file |>
  drive_share(
    role = "writer",
    type = "user",
    emailAddress = "carol@example.com",
    emailMessage = "Would appreciate your feedback on this!"
  )

# Let anyone read the file
file <- file |>
  drive_share(role = "reader", type = "anyone")
# Single-purpose wrapper function for this
drive_share_anyone(file)

# Clean up
drive_rm(file)
```

drive_token *Produce configured token*

Description

For internal use or for those programming around the Drive API. Returns a token pre-processed with `httr::config()`. Most users do not need to handle tokens "by hand" or, even if they need some control, `drive_auth()` is what they need. If there is no current token, `drive_auth()` is called to either load from cache or initiate OAuth2.0 flow. If auth has been deactivated via `drive_deauth()`, `drive_token()` returns `NULL`.

Usage

```
drive_token()
```

Value

A `request` object (an S3 class provided by httr).

See Also

Other low-level API functions: `drive_has_token()`, `request_generate()`, `request_make()`

Examples

```
req <- request_generate(
  "drive.files.get",
  list(fileId = "abc"),
  token = drive_token()
)
req
```

drive_token_write *Write/read a googledrive user token*

Description

[Experimental]

This pair of functions writes an OAuth2 user token to file and reads it back in. This is rarely necessary when working in your primary, interactive computing environment. In that setting, it is recommended to lean into the automatic token caching built-in to googledrive / gargle. However, when preparing a user token for use elsewhere, such as in CI or in a deployed data product, it can be useful to take the full control granted by `drive_token_write()` and `drive_token_read()`.

Below is an outline of the intended workflow, but you will need to fill in particulars, such as filepaths and environment variables:

- Do auth in your primary, interactive environment as the target user, with the desired OAuth client and scopes.

```
drive_auth("jane@example.com", cache = FALSE)
```

- Confirm you are logged in as the intended user:

```
drive_user()
```

- Write the current token to file:

```
drive_token_write(
  path = "path/to/googledrive-token.rds",
  key = "GOOGLEDRIVE_TOKEN_KEY"
)
```

- In the deployed, non-interactive setting, read the token from file and tell googledrive to use it:

```
drive_auth(token = drive_token_read(
  path = "path/to/googledrive-token.rds",
  key = "GOOGLEDRIVE_TOKEN_KEY"
))
```

Usage

```
drive_token_write(
  token = drive_token(),
  path = "googledrive-token.rds",
  key = NULL
)

drive_token_read(path = "googledrive-token.rds", key = NULL)
```

Arguments

token	A token with class Token2.0 or an object of httr's class request, i.e. a token that has been prepared with httr::config() and has a Token2.0 in the auth_token component.
path	The path to write to (drive_token_write()) or to read from (drive_token_read()).
key	Encryption key, as implemented by httr2's secret functions. If absent, a built-in key is used. If supplied, the key should usually be the name of an environment variable whose value was generated with gargle::secret_make_key() (which is a copy of httr2::secret_make_key()). The key argument of drive_token_read() must match the key used in drive_token_write().

Security

drive_token_write() and drive_token_read() have a more security-oriented implementation than the default token caching strategy. OAuth2 user tokens are somewhat opaque by definition, because they aren't written to file in a particularly transparent format. However, drive_token_write() always applies some additional obfuscation to make such credentials even more resilient against

scraping by an automated tool. However, a knowledgeable R programmer could decode the credential with some effort. The default behaviour of `drive_token_write()` (called without `key`) is suitable for tokens stored in a relatively secure place, such as on Posit Connect within your organization.

To prepare a stored credential for exposure in a more public setting, such as on GitHub or CRAN, you must actually encrypt it, using a `key` known only to you. You must make the encryption `key` available via a secure environment variable in any setting where you wish to decrypt and use the token, such as on GitHub Actions.

`drive_trash` *Move Drive files to or from trash*

Description

Move Drive files to or from trash

Usage

```
drive_trash(file)

drive_untrash(file)
```

Arguments

`file`	Something that identifies the file(s) of interest on your Google Drive. Can be a character vector of names/paths, a character vector of file ids or URLs marked with `as_id()`, or a `dribble`.

Value

An object of class `dribble`, a tibble with one row per file.

Examples

```
# Create a file and put it in the trash.
file <- drive_example_remote("chicken.txt") |>
  drive_cp("chicken-trash.txt")
drive_trash("chicken-trash.txt")

# Confirm it's in the trash
drive_find(trashed = TRUE)

# Remove it from the trash and confirm
drive_untrash("chicken-trash.txt")
drive_find(trashed = TRUE)

# Clean up
drive_rm("chicken-trash.txt")
```

drive_update *Update an existing Drive file*

Description

Update an existing Drive file id with new content ("media" in Drive API-speak), new metadata, or both. To create a new file or update existing, depending on whether the Drive file already exists, see `drive_put()`.

Usage

```
drive_update(file, media = NULL, ...)
```

Arguments

`file`	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with `as_id()`, or a `dribble`.
`media`	Character, path to the local file to upload.
`...`	Named parameters to pass along to the Drive API. Has dynamic dots semantics. You can affect the metadata of the target file by specifying properties of the Files resource via `....` Read the "Request body" section of the Drive API docs for the associated endpoint to learn about relevant parameters.

Value

An object of class `dribble`, a tibble with one row per file.

See Also

Wraps the `files.update` endpoint:

- https://developers.google.com/drive/api/v3/reference/files/update

This function supports media upload:

- https://developers.google.com/drive/api/v3/manage-uploads

Examples

```
# Create a new file, so we can update it
x <- drive_example_remote("chicken.csv") |>
  drive_cp()

# Update the file with new media
x <- x |>
  drive_update(drive_example_local("chicken.txt"))

# Update the file with new metadata.
# Notice here `name` is not an argument of `drive_update()`, we are passing
```

```
# this to the API via the `...``
x <- x |>
  drive_update(name = "CHICKENS!")

# Update the file with new media AND new metadata
x <- x |>
  drive_update(
    drive_example_local("chicken.txt"),
    name = "chicken-poem-again.txt"
  )

# Clean up
drive_rm(x)
```

`drive_upload` *Upload into a new Drive file*

Description

Uploads a local file into a new Drive file. To update the content or metadata of an existing Drive file, use `drive_update()`. To upload or update, depending on whether the Drive file already exists, see `drive_put()`.

Usage

```
drive_upload(media, path = NULL, name = NULL, type = NULL, ..., overwrite = NA)
```

Arguments

`media`	Character, path to the local file to upload.
`path`	Specifies target destination for the new file on Google Drive. Can be an actual path (character), a file id marked with `as_id()`, or a `dribble`. If `path` is a shortcut to a folder, it is automatically resolved to its target folder. If `path` is given as a path (as opposed to a `dribble` or an id), it is best to explicitly indicate if it's a folder by including a trailing slash, since it cannot always be worked out from the context of the call. By default, the file is created in the current user's "My Drive" root folder.
`name`	Character, new file name if not specified as part of `path`. This will force `path` to be interpreted as a folder, even if it is character and lacks a trailing slash. Defaults to the file's local name.
`type`	Character. If `type = NULL`, a MIME type is automatically determined from the file extension, if possible. If the source file is of a suitable type, you can request conversion to Google Doc, Sheet or Slides by setting `type` to `document`, `spreadsheet`, or `presentation`, respectively. All non-`NULL` values for `type` are pre-processed with `drive_mime_type()`.

`...` Named parameters to pass along to the Drive API. Has dynamic dots semantics. You can affect the metadata of the target file by specifying properties of the Files resource via `...`. Read the "Request body" section of the Drive API docs for the associated endpoint to learn about relevant parameters.

`overwrite` Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in `drive_get()`.

- `NA` (default): Just do the operation, even if it results in multiple files with the same filepath.
- `TRUE`: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use `drive_update()` or `drive_put()`.
- `FALSE`: Error if there is any pre-existing file at the filepath.

Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class `dribble`, a tibble with one row per file.

See Also

Wraps the `files.create` endpoint:

- https://developers.google.com/drive/api/v3/reference/files/create

MIME types that can be converted to native Google formats:

- https://developers.google.com/drive/api/v3/manage-uploads#import_to_google_docs_types

Examples

```
# upload a csv file
chicken_csv <- drive_example_local("chicken.csv") |>
  drive_upload("chicken-upload.csv")

# or convert it to a Google Sheet
chicken_sheet <- drive_example_local("chicken.csv") |>
  drive_upload(
    name = "chicken-sheet-upload.csv",
    type = "spreadsheet"
  )

# check out the new Sheet!
drive_browse(chicken_sheet)
```

```
# Clean up
drive_find("chicken.*upload") |> drive_rm()

# Upload a file and, at the same time, star it
chicken <- drive_example_local("chicken.jpg") |>
  drive_upload(starred = "true")

# Is it really starred? YES
purrr::pluck(chicken, "drive_resource", 1, "starred")

# Clean up
drive_rm(chicken)

# `overwrite = FALSE` errors if something already exists at target filepath
# THIS WILL ERROR!
drive_create("name-squatter-upload")
drive_example_local("chicken.jpg") |>
  drive_upload(
    name = "name-squatter-upload",
    overwrite = FALSE
  )

# `overwrite = TRUE` moves the existing item to trash, then proceeds
chicken <- drive_example_local("chicken.jpg") |>
  drive_upload(
    name = "name-squatter-upload",
    overwrite = TRUE
  )

# Clean up
drive_rm(chicken)

## Not run:
# Upload to a shared drive:
#   * Shared drives are only available if your account is associated with a
#     Google Workspace
#   * The shared drive (or shared-drive-hosted folder) MUST be captured as a
#     dribble first and provided via `path`
sd <- shared_drive_get("Marketing")
drive_upload("fascinating.csv", path = sd)

## End(Not run)
```

---

drive_user *Get info on current user*

Description

Reveals information about the user associated with the current token. This is a thin wrapper around `drive_about()` that just extracts the most useful information (the information on current user) and prints it nicely.

Usage

```
drive_user()
```

Value

A list of class `drive_user`.

See Also

Wraps the `about.get` endpoint:

- https://developers.google.com/drive/api/v3/reference/about/get

Examples

```
drive_user()

# more info is returned than is printed
user <- drive_user()
str(user)
```

`googledrive-configuration`

googledrive configuration

Description

Some aspects of googledrive behaviour can be controlled via an option.

Usage

```
local_drive_quiet(env = parent.frame())

with_drive_quiet(code)
```

Arguments

`env`	The environment to use for scoping
`code`	Code to execute quietly

Auth

Read about googledrive's main auth function, `drive_auth()`. It is powered by the gargle package, which consults several options:

- Default Google user or, more precisely, `email`: see `gargle::gargle_oauth_email()`
- Whether or where to cache OAuth tokens: see `gargle::gargle_oauth_cache()`
- Whether to prefer "out-of-band" auth: see `gargle::gargle_oob_default()`
- Application Default Credentials: see `gargle::credentials_app_default()`

Messages

The `googledrive_quiet` option can be used to suppress messages from googledrive. By default, googledrive always messages, i.e. it is *not* quiet.

Set `googledrive_quiet` to `TRUE` to suppress messages, by one of these means, in order of decreasing scope:

- Put `options(googledrive_quiet = TRUE)` in a start-up file, such as `.Rprofile`, or at the top of your R script
- Use `local_drive_quiet()` to silence googledrive in a specific scope

```
foo <- function() {
  ...
  local_drive_quiet()
  drive_this(...)
  drive_that(...)
  ...
}
```

- Use `with_drive_quiet()` to run a small bit of code silently

```
with_drive_quiet(
  drive_something(...)
)
```

`local_drive_quiet()` and `with_drive_quiet()` follow the conventions of the the withr package (https://withr.r-lib.org).

Examples

```
# message: "Created Drive file"
(x <- drive_create("drive-quiet-demo", type = "document"))

# message: "File updated"
x <- drive_update(x, starred = TRUE)
drive_reveal(x, "starred")

# suppress messages for a small amount of code
with_drive_quiet(
  x <- drive_update(x, name = "drive-quiet-works")
)
```

```
x$name

# message: "File updated"
x <- drive_update(x, media = drive_example_local("chicken.txt"))

# suppress messages within a specific scope, e.g. function
unstar <- function(y) {
  local_drive_quiet()
  drive_update(y, starred = FALSE)
}
x <- unstar(x)
drive_reveal(x, "starred")

# Clean up
drive_rm(x)
```

`request_generate` *Build a request for the Google Drive API*

Description

Build a request, using knowledge of the Drive v3 API from its Discovery Document (`https://www.googleapis.com/disco`
Most users should, instead, use higher-level wrappers that facilitate common tasks, such as uploading or downloading Drive files. The functions here are intended for internal use and for programming around the Drive API.

`request_generate()` lets you provide the bare minimum of input. It takes a nickname for an endpoint and:

- Uses the API spec to look up the `path`, `method`, and base URL.
- Checks `params` for validity and completeness with respect to the endpoint. Separates parameters into those destined for the body, the query, and URL endpoint substitution (which is also enacted).
- Adds an API key to the query if and only if `token = NULL`.
- Adds `supportsAllDrives = TRUE` to the query if the endpoint requires.

Usage

```
request_generate(
  endpoint = character(),
  params = list(),
  key = NULL,
  token = drive_token()
)
```

Arguments

endpoint	Character. Nickname for one of the selected Drive v3 API endpoints built into googledrive. Learn more in drive_endpoints().
params	Named list. Parameters destined for endpoint URL substitution, the query, or the body.
key	API key. Needed for requests that don't contain a token. The need for an API key in the absence of a token is explained in Google's document "Credentials, access, security, and identity" (https://support.google.com/googleapi/answer/6158857?hl=en&ref In order of precedence, these sources are consulted: the formal key argument, a key parameter in params, a user-configured API key fetched via drive_api_key(), a built-in key shipped with googledrive. See drive_auth_configure() for details on a user-configured key.
token	Drive token. Set to NULL to suppress the inclusion of a token. Note that, if auth has been de-activated via drive_deauth(), drive_token() will actually return NULL.

Value

list()
Components are method, path, query, body, token, and url, suitable as input for request_make().

See Also

gargle::request_develop(), gargle::request_build()

Other low-level API functions: drive_has_token(), drive_token(), request_make()

Examples

```
req <- request_generate(
  "drive.files.get",
  list(fileId = "abc"),
  token = drive_token()
)
req
```

---

request_make *Make a request for the Google Drive v3 API*

Description

Low-level functions to execute one or more Drive API requests and, perhaps, process the response(s). Most users should, instead, use higher-level wrappers that facilitate common tasks, such as uploading or downloading Drive files. The functions here are intended for internal use and for programming around the Drive API. Three functions are documented here:

- `request_make()` does the bare minimum: calls `gargle::request_retry()`, only adding the googledrive user agent. Typically the input is created with `request_generate()` and the output is processed with `gargle::response_process()`.
- `do_request()` is simply `gargle::response_process(request_make(x, ...))`. It exists only because we had to make `do_paginated_request()` and it felt weird to not make the equivalent for a single request.
- `do_paginated_request()` executes the input request **with page traversal**. It is impossible to separate paginated requests into a "make request" step and a "process request" step, because the token for the next page must be extracted from the content of the current page. Therefore this function does both and returns a list of processed responses, one per page.

Usage

```
request_make(x, ...)

do_request(x, ...)

do_paginated_request(x, ..., n_max = Inf, n = function(res) 1)
```

Arguments

`x`	List, holding the components for an HTTP request, presumably created with `request_generate()` Should contain the `method`, `url`, `body`, and `token`.
`...`	Optional arguments passed through to the HTTP method.
`n_max`	Maximum number of items to return. Defaults to `Inf`, i.e. there is no limit and we keep making requests until we get all items.
`n`	Function that computes the number of items in one response or page. The default function always returns `1` and therefore treats each page as an item. If you know more about the structure of the response, you can pass another function to count and threshold, for example, the number of files or comments.

Value

`request_make()`: Object of class `response` from httr.

`do_request()`: List representing the content returned by a single request.

`do_paginated_request()`: List of lists, representing the returned content, one component per page.

See Also

Other low-level API functions: `drive_has_token()`, `drive_token()`, `request_generate()`

Examples

```
## Not run:
# build a request for an endpoint that is:
#   * paginated
#   * NOT privileged in googledrive, i.e. not covered by request_generate()
```

```
# "comments" are a great example
# https://developers.google.com/drive/v3/reference/comments
#
# Practice with a target file with > 2 comments
# Note that we request 2 items (comments) per page
req <- gargle::request_build(
  path = "drive/v3/files/{fileId}/comments",
  method = "GET",
  params = list(
    fileId = "your-file-id-goes-here",
    fields = "*",
    pageSize = 2
  ),
  token = googledrive::drive_token()
)
# make the paginated request, but cap it at 1 page
# should get back exactly two comments
do_paginated_request(req, n_max = 1)

## End(Not run)
```

`shared_drive_create` *Create a new shared drive*

Description

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives. Learn more about shared drives.

Usage

```
shared_drive_create(name)
```

Arguments

`name`	Character. Name of the new shared drive. Must be non-empty and not entirely whitespace.

Value

An object of class `dribble`, a tibble with one row per shared drive.

See Also

Wraps the `drives.create` endpoint:

- https://developers.google.com/drive/api/v3/reference/drives/create

Examples

```
## Not run:
shared_drive_create("my-awesome-shared-drive")

# Clean up
shared_drive_rm("my-awesome-shared-drive")

## End(Not run)
```

`shared_drive_find` *Find shared drives*

Description

This is the closest googledrive function to what you get from visiting https://drive.google.com and clicking "Shared drives".

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives. Learn more about shared drives.

Usage

```
shared_drive_find(pattern = NULL, n_max = Inf, ...)
```

Arguments

`pattern`	Character. If provided, only the items whose names match this regular expression are returned. This is implemented locally on the results returned by the API.
`n_max`	Integer. An upper bound on the number of items to return. This applies to the results requested from the API, which may be further filtered locally, via the `pattern` argument.
`...`	Other parameters to pass along in the request, such as `pageSize` or `useDomainAdminAccess`.

Value

An object of class `dribble`, a tibble with one row per shared drive.

See Also

Wraps the `drives.list` endpoint:

- https://developers.google.com/drive/api/v3/reference/drives/list

Examples

```
## Not run:
shared_drive_find()

## End(Not run)
```

`shared_drive_get` *Get shared drives by name or id*

Description

Retrieve metadata for shared drives specified by name or id. Note that Google Drive does NOT behave like your local file system:

- You can get zero, one, or more shared drives back for each name! Shared drive names need not be unique.

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives. Learn more about shared drives.

Usage

```
shared_drive_get(name = NULL, id = NULL)
```

Arguments

`name`	Character vector of names. A character vector marked with `as_id()` is treated as if it was provided via the `id` argument.
`id`	Character vector of shared drive ids or URLs (it is first processed with `as_id()`). If both `name` and `id` are non-`NULL`, `id` is silently ignored.

Value

An object of class `dribble`, a tibble with one row per shared drive.

Examples

```
## Not run:
shared_drive_get("my-awesome-shared-drive")
shared_drive_get(c("apple", "orange", "banana"))
shared_drive_get(as_id("KCmiHLXUk9PVA-0AJNG"))
shared_drive_get(as_id("https://drive.google.com/drive/u/0/folders/KCmiHLXUk9PVA-0AJNG"))
shared_drive_get(id = "KCmiHLXUk9PVA-0AJNG")
shared_drive_get(id = "https://drive.google.com/drive/u/0/folders/KCmiHLXUk9PVA-0AJNG")

## End(Not run)
```

shared_drive_rm *Delete shared drives*

Description

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives. Learn more about shared drives.

Usage

```
shared_drive_rm(drive = NULL)
```

Arguments

drive	Anything that identifies the shared drive(s) of interest. Can be a character vector of names, a character vector of file ids or URLs marked with as_id(), or a dribble consisting only of shared drives.

Value

Logical vector, indicating whether the delete succeeded.

See Also

Wraps the drives.delete endpoint:

- https://developers.google.com/drive/api/v3/reference/drives/delete

Examples

```
## Not run:
# Create shared drives to remove in various ways
shared_drive_create("testdrive-01")
sd02 <- shared_drive_create("testdrive-02")
shared_drive_create("testdrive-03")
sd04 <- shared_drive_create("testdrive-04")

# remove by name
shared_drive_rm("testdrive-01")
# remove by id
shared_drive_rm(as_id(sd02))
# remove by URL (or, rather, id found in URL)
shared_drive_rm(as_id("https://drive.google.com/drive/u/0/folders/Q5DqUk9PVA"))
# remove by dribble
shared_drive_rm(sd04)

## End(Not run)
```

`shared_drive_update` *Update a shared drive*

Description

Update the metadata of an existing shared drive, e.g. its background image or theme.

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives. Learn more about shared drives.

Usage

```
shared_drive_update(shared_drive, ...)
```

Arguments

`shared_drive`	Anything that identifies one specific shared drive: its name, its id or URL marked with `as_id()`, or a `dribble`. The value provided to `shared_drive` is pre-processed with `as_shared_drive()`. Read more about shared drives.
`...`	Properties to set in `name = value` form. See the "Request body" section of the Drive API docs for this endpoint.

Value

An object of class `dribble`, a tibble with one row per shared drive.

See Also

Wraps the `drives.update` endpoint:

- https://developers.google.com/drive/api/v3/reference/drives/update

Examples

```
## Not run:
# create a shared drive
sd <- shared_drive_create("I love themes!")

# see the themes available to you
themes <- drive_about()$driveThemes
purrr::map_chr(themes, "id")

# cycle through various themes for this shared drive
sd <- shared_drive_update(sd, themeId = "bok_choy")
sd <- shared_drive_update(sd, themeId = "cocktails")

# Clean up
shared_drive_rm(sd)

## End(Not run)
```

shared_drives *Access shared drives*

Description

A shared drive supports files owned by an organization rather than an individual user. Shared drives follow different sharing and ownership models from a specific user's "My Drive". Shared drives are the successors to the earlier concept of Team Drives.

How to capture a shared drive or files/folders that live on a shared drive for downstream use:

- `shared_drive_find()` and `shared_drive_get()` return a `dribble` with metadata on shared drives themselves. You will need this in order to use a shared drive in certain file operations. For example, you can specify a shared drive as the parent folder via the `path` argument for upload, move, copy, etc. In that context, the id of a shared drive functions like the id of its top-level or root folder.
- `drive_find()` and `drive_get()` return a `dribble` with metadata on files, including folders. Both can be directed to search for files on shared drives using the optional arguments `shared_drive` or `corpus` (documented below).

Regard the functions mentioned above as the official "port of entry" for working with shared drives. Use these functions to capture your target(s) in a `dribble` to pass along to other googledrive functions. The flexibility to refer to files by name or path does not apply as broadly to shared drives. While it's always a good idea to get things into a `dribble` early, for shared drives it's often required.

Specific shared drive

To search one specific shared drive, pass its name, marked id, or `dribble` to `shared_drive` somewhere in the call, like so:

```
drive_find(..., shared_drive = "i_am_a_shared_drive_name")
drive_find(..., shared_drive = as_id("i_am_a_shared_drive_id"))
drive_find(..., shared_drive = i_am_a_shared_drive_dribble)
```

The value provided to `shared_drive` is pre-processed with `as_shared_drive()`.

Other collections

To search other collections, pass the `corpus` parameter somewhere in the call, like so:

```
drive_find(..., corpus = "user")
drive_find(..., corpus = "allDrives")
drive_find(..., corpus = "domain")
```

Possible values of `corpus` and what they mean:

- `"user"`: Queries files that the user has accessed, including both shared drive and My Drive files.

- "drive": Queries all items in the shared drive specified via shared_drive. googledrive automatically fills this in whenever shared_drive is not NULL.
- "allDrives": Queries files that the user has accessed and all shared drives in which they are a member. Note that the response may include incompleteSearch : true, indicating that some corpora were not searched for this request (currently, googledrive does not surface this). Prefer "user" or "drive" to "allDrives" for efficiency.
- "domain": Queries files that are shared to the domain, including both shared drive and My Drive files.

Google blogs and docs

Here is some of the best official Google reading about shared drives:

- Team Drives is being renamed to shared drives from Google Workspace blog
- Upcoming changes to the Google Drive API and Google Picker API from the Google Cloud blog
- https://developers.google.com/drive/api/v3/about-shareddrives
- https://developers.google.com/drive/api/v3/shared-drives-diffs
- Get started with shared drives: https://support.google.com/a/users/answer/9310351 from Google Workspace Learning Center
- Best practices for shared drives: https://support.google.com/a/users/answer/9310156 from Google Workspace Learning Center

API docs

googledrive implements shared drive support as outlined here:

- https://developers.google.com/drive/api/v3/enable-shareddrives

Users shouldn't need to know any of this, but here are details for the curious. The extra information needed to search shared drives consists of the following query parameters:

- corpora: Where to search? Formed from googledrive's corpus argument.
- driveId: The id of a specific shared drive. Only allowed – and also absolutely required – when corpora = "drive". When user specifies a shared_drive, googledrive sends its id and also infers that corpora should be set to "drive".
- includeItemsFromAllDrives: Do you want to see shared drive items? Obviously, this should be TRUE and googledrive sends this whenever shared drive parameters are detected.
- supportsAllDrives: Does the sending application (googledrive, in this case) know about shared drive? Obviously, this should be TRUE and googledrive sends it for all applicable endpoints, all the time.

shortcut_create *Create a shortcut to a Drive file*

Description

Creates a shortcut to the target Drive `file`, which could be a folder. A Drive shortcut functions like a symbolic or "soft" link and is primarily useful for creating a specific Drive user experience in the browser, i.e. to make a Drive file or folder appear in more than 1 place. Shortcuts are a relatively new feature in Drive; they were introduced when Drive stopped allowing a file to have more than 1 parent folder.

Usage

```
shortcut_create(file, path = NULL, name = NULL, overwrite = NA)
```

Arguments

`file`	Something that identifies the file of interest on your Google Drive. Can be a name or path, a file id or URL marked with `as_id()`, or a `dribble`.
`path`	Target destination for the new shortcut, i.e. a folder or a shared drive. Can be given as an actual path (character), a file id or URL marked with `as_id()`, or a `dribble`. Defaults to your "My Drive" root folder. If `path` is a shortcut to a folder, it is automatically resolved to its target folder.
`name`	Character, new shortcut name if not specified as part of `path`. This will force `path` to be interpreted as a folder, even if it is character and lacks a trailing slash. By default, the shortcut starts out with the same name as the target `file`. As a consequence, if you want to use `overwrite = TRUE` or `overwrite = FALSE`, you **must** explicitly specify the shortcut's `name`.
`overwrite`	Logical, indicating whether to check for a pre-existing file at the targeted "filepath". The quotes around "filepath" refer to the fact that Drive does not impose a 1-to-1 relationship between filepaths and files, like a typical file system; read more about that in `drive_get()`. • `NA` (default): Just do the operation, even if it results in multiple files with the same filepath. • `TRUE`: Check for a pre-existing file at the filepath. If there is zero or one, move a pre-existing file to the trash, then carry on. Note that the new file does not inherit any properties from the old one, such as sharing or publishing settings. It will have a new file ID. An error is thrown if two or more pre-existing files are found. If you need to keep the file ID (and its sharing/publishing settings), use `drive_update()` or `drive_put()`. • `FALSE`: Error if there is any pre-existing file at the filepath. Note that existence checks, based on filepath, are expensive operations, i.e. they require additional API calls.

Value

An object of class dribble, a tibble with one row per file.

See Also

- https://developers.google.com/drive/api/v3/shortcuts

Examples

```
# Target one of the official example files
(src_file <- drive_example_remote("chicken_sheet"))

# Create a shortcut in the default location with the default name
sc1 <- shortcut_create(src_file)
# This shortcut could now be moved, renamed, etc.

# Create a shortcut in the default location with a custom name
sc2 <- src_file |>
  shortcut_create(name = "chicken_sheet_second_shortcut")

# Create a folder, then put a shortcut there, with default name
folder <- drive_mkdir("chicken_sheet_shortcut_folder")
sc3 <- src_file |>
  shortcut_create(folder)

# Look at all these shortcuts
(dat <- drive_find("chicken_sheet", type = "shortcut"))

# Confirm the shortcuts all target the original file
dat <- dat |>
  drive_reveal("shortcut_details")
purrr::map_chr(dat$shortcut_details, "targetId")
as_id(src_file)

# Clean up
drive_rm(sc1, sc2, sc3, folder)
```

---

shortcut_resolve *Resolve shortcuts to their targets*

Description

Retrieves the metadata for the Drive file that a shortcut refers to, i.e. the shortcut's target. The returned dribble has the usual columns (name, id, drive_resource), which refer to the target. It will also include the columns name_shortcut and id_shortcut, which refer to the original shortcut. There are 3 possible scenarios:

1. file is a shortcut and user can drive_get() the target. All is simple and well.

2. `file` is a shortcut, but `drive_get()` fails for the target. This can happen if the user can see the shortcut, but does not have read access to the target. It can also happen if the target has been trashed or deleted. In such cases, all of the target's metadata, except for `id`, will be missing. Call `drive_get()` on a problematic `id` to see the specific error.
3. `file` is not a shortcut. `name_shortcut` and `id_shortcut` will both be `NA`.

Usage

```
shortcut_resolve(file)
```

Arguments

`file`	Something that identifies the file(s) of interest on your Google Drive. Can be a character vector of names/paths, a character vector of file ids or URLs marked with `as_id()`, or a `dribble`.

Value

An object of class `dribble`, a tibble with one row per file. Extra columns `name_shortcut` and `id_shortcut` refer to the original shortcut.

Examples

```
# Create a file to make a shortcut to
file <- drive_example_remote("chicken_sheet") |>
  drive_cp(name = "chicken-sheet-for-shortcut")

# Create a shortcut
sc1 <- file |>
  shortcut_create(name = "shortcut-1")

# Create a second shortcut by copying the first
sc1 <- sc1 |>
  drive_cp(name = "shortcut-2")

# Get the shortcuts
(sc_dat <- drive_find("-[12]$", type = "shortcut"))

# Resolve them
(resolved <- shortcut_resolve(sc_dat))

resolved$id
file$id

# Delete the target file
drive_rm(file)

# (Try to) resolve the shortcuts again
shortcut_resolve(sc_dat)
# No error, but resolution is unsuccessful due to non-existent target
```

```
# Clean up
drive_rm(sc_dat)
```

Index

∗ **auth functions**
drive_auth, 8
drive_auth_configure, 11
drive_deauth, 17
drive_scopes, 45
∗ **low-level API functions**
drive_has_token, 28
drive_token, 48
request_generate, 57
request_make, 58

abort(), *6*
as_dribble, 3
as_dribble(), *5*, *44*
as_id *(*drive_id*)*, 29
as_id(), *3, 4*, *12, 13*, *15*, *18*, *23*, *26*, *30–32*, *34*, *36*, *38–41*, *44*, *46*, *50–52*, *62–64*, *67*, *69*
as_shared_drive, 4
as_shared_drive(), *23*, *26*, *64, 65*

confirm_dribble *(*dribble-checks*)*, 5
confirm_single_file *(*dribble-checks*)*, 5
confirm_some_files *(*dribble-checks*)*, 5

do_paginated_request *(*request_make*)*, 58
do_request *(*request_make*)*, 58
dribble, *3, 4*, 5, *5, 6*, *12–16*, *18*, *23, 24*, *26, 27*, *29–36*, *38–44*, *46, 47*, *50–53*, *60–65*, *67–69*
dribble-checks, 5
drive_about, 7
drive_about(), *55*
drive_api_key *(*drive_auth_configure*)*, 11
drive_api_key(), *17*, *58*
drive_auth, 8
drive_auth(), *11*, *17*, *45*, *48*, *56*
drive_auth_configure, 11
drive_auth_configure(), *10*, *17*, *45*, *58*
drive_browse, 12
drive_cp, 13
drive_cp(), *3*
drive_create, 15
drive_deauth, 17
drive_deauth(), *10, 11*, *45*, *48*, *58*
drive_download, 18
drive_download(), *39*
drive_empty_trash, 19
drive_endpoint *(*drive_endpoints*)*, 19
drive_endpoints, 19
drive_endpoints(), *58*
drive_example_local *(*drive_examples*)*, 20
drive_example_remote *(*drive_examples*)*, 20
drive_examples, 20
drive_examples_local *(*drive_examples*)*, 20
drive_examples_remote *(*drive_examples*)*, 20
drive_extension, 21
drive_fields, 22
drive_find, 23
drive_find(), *27*, *30, 31*, *65*
drive_get, 26
drive_get(), *13*, *15*, *23*, *33, 34*, *40*, *53*, *65*, *67–69*
drive_has_token, 28
drive_has_token(), *48*, *58, 59*
drive_id, 29
drive_link, 30
drive_link(), *12*
drive_ls, 30
drive_ls(), *23*, *27*
drive_mime_type, 31
drive_mime_type(), *15*, *18*, *23, 24*, *38, 39*, *52*
drive_mkdir, 32
drive_mkdir(), *3*, *15*
drive_mv, 34
drive_mv(), *3*, *40*

drive_oauth_client
(drive_auth_configure), 11
drive_publish, 36
drive_put, 37
drive_put(), 51, 52
drive_read_raw (drive_read_string), 39
drive_read_string, 39
drive_rename, 40
drive_reveal, 41
drive_reveal(), 36, 46
drive_rm, 44
drive_scopes, 45
drive_scopes(), 9–11, 17, 45
drive_share, 46
drive_share_anyone (drive_share), 46
drive_token, 48
drive_token(), 29, 58, 59
drive_token_read (drive_token_write), 48
drive_token_write, 48
drive_trash, 50
drive_trash(), 44
drive_unpublish (drive_publish), 36
drive_untrash (drive_trash), 50
drive_update, 51
drive_update(), 32, 37, 52
drive_upload, 52
drive_upload(), 3, 15, 37
drive_user, 54
drive_user(), 7
dynamic dots, 13, 15, 33, 38, 51, 53

expose(), 32

gargle::AuthState, 11
gargle::credentials_app_default(), 56
gargle::gargle_oauth_cache(), 56
gargle::gargle_oauth_client_from_json(),
11
gargle::gargle_oauth_email(), 56
gargle::gargle_oob_default(), 56
gargle::gargle_options, 10
gargle::request_build(), 58
gargle::request_develop(), 58
gargle::request_retry(), 59
gargle::response_process(), 59
gargle::token_fetch(), 8, 10
gargle_oauth_cache(), 9
gargle_oauth_client_type(), 9
gargle_oauth_email(), 8
gargle_oob_default(), 9
googledrive-configuration, 55

httr, 48, 59
httr::config(), 9, 48, 49
httr::content(), 39

is_dribble (dribble-checks), 5
is_folder (dribble-checks), 5
is_folder_shortcut (dribble-checks), 5
is_mine (dribble-checks), 5
is_native (dribble-checks), 5
is_parental (dribble-checks), 5
is_shared_drive (dribble-checks), 5
is_shortcut (dribble-checks), 5

jsonlite::fromJSON(), 8, 11

local_drive_quiet
(googledrive-configuration), 55

no_file (dribble-checks), 5

POSIXct, 41
prep_fields (drive_fields), 22

raw(), 39
request_generate, 57
request_generate(), 19, 29, 48, 59
request_make, 58
request_make(), 29, 48, 58

shared drives, 4, 23, 24, 26, 27, 60–64
shared_drive_create, 60
shared_drive_find, 61
shared_drive_find(), 65
shared_drive_get, 62
shared_drive_get(), 65
shared_drive_rm, 63
shared_drive_update, 64
shared_drives, 65
shortcut_create, 67
shortcut_resolve, 68
single_file (dribble-checks), 5
some_files (dribble-checks), 5

tibble, 5
Token2.0, 9, 49

with_drive_quiet
(googledrive-configuration), 55