Package 'AnVILGCP'

November 11, 2026

Title The GCP R Client for the AnVIL

Version 1.6.2

Date 2026-04-24

Description The package provides a set of functions to interact with the Google Cloud Platform (GCP) services on the AnVIL platform. The package is designed to use the API calls from the AnVIL package. It coordinates AnVIL workspace functionality with native GCP tools.

biocViews Software, Infrastructure, ThirdPartyClient, DataImport

Depends R (>= 4.5.0)

Imports AnVILBase, BiocBaseUtils, dplyr, GCPtools (>= 0.99.4), httr, jsonlite, methods, rlang, stats, tibble, tidyr, utils

Suggests AnVIL, BiocStyle, knitr, rmarkdown, testthat, withr

VignetteBuilder knitr

BugReports https://github.com/Bioconductor/AnVILGCP/issues

URL https://github.com/Bioconductor/AnVILGCP

License Artistic-2.0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Collate 'av.R' 'avdata.R' 'gcp-class.R' 'avnotebooks-methods.R' 'avtable-methods.R' 'avworkflow-methods.R' 'avworkflow.R' 'avworkflow_configuration.R' 'avworkspace-methods.R' 'drs.R' 'gcloud.R' 'gcloud_sdk.R' 'gcp-methods.R' 'gsutil.R' 'has_avworkspace-methods.R' 'localize.R' 'utilities.R'

git_url https://git.bioconductor.org/packages/AnVILGCP

git_branch RELEASE_3_23

git_last_commit 6d610c7

git_last_commit_date 2026-05-26

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Marcel Ramos [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3242-0582>),
Nitesh Turaga [aut],
Martin Morgan [aut] (ORCID: <https://orcid.org/0000-0002-5874-8148>)

Maintainer Marcel Ramos <marcel.ramos@sph.cuny.edu>

Contents

av 2
avdata 7
avnotebooks-methods 8
avtable-methods 9
avworkflow-methods 14
avworkflows 15
avworkflow_configurations 19
avworkspace-methods 23
drs 25
gcloud-defunct 27
gcp-class 28
gcp-methods 29
gsutil-deprecated 32
has_avworkspace-methods 35
localize 35

Index **37**

---

av *Miscellaneous functions for interacting with AnVIL tables and files*

Description

`avtable_import_status()` queries for the status of an 'asynchronous' table import.

`avfiles_ls()` returns the paths of files in the workspace bucket. `avfiles_backup()` copies files from the compute node file system to the workspace bucket. `avfiles_restore()` copies files from the workspace bucket to the compute node file system. `avfiles_rm()` removes files or directories from the workspace bucket.

`avruntimes()` returns a tibble containing information about runtimes (notebooks or RStudio instances, for example) that the current user has access to.

`avruntime()` returns a tibble with the runtimes associated with a particular google project and account number; usually there is a single runtime satisfiying these criteria, and it is the runtime active in AnVIL.

'avdisks()' returns a tibble containing information about persistent disks associatd with the current user.

Usage

```
avtable_paged(
  table,
  n = Inf,
  page = 1L,
  pageSize = 1000L,
  sortField = "name",
  sortDirection = c("asc", "desc"),
  filterTerms = character(),
  filterOperator = c("and", "or"),
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  na = c("", "NA")
)

avtable_import_status(
  job_status,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_ls(
  path = "",
  full_names = FALSE,
  recursive = FALSE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_backup(
  source,
  destination = "",
  recursive = FALSE,
  parallel = TRUE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_restore(
  source,
  destination = ".",
  recursive = FALSE,
  parallel = TRUE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avfiles_rm(
```

```
  source,
  recursive = FALSE,
  parallel = TRUE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avruntimes()

avruntime(
  project = GCPtools::gcloud_project(),
  account = GCPtools::gcloud_account()
)

avdisks()
```

Arguments

table	character(1) table name as returned by, e.g., avtables().
n	numeric(1) maximum number of rows to return
page	integer(1) first page of iteration
pageSize	integer(1) number of records per page. Generally, larger page sizes are more efficient.
sortField	character(1) field used to sort records when determining page order. Default is the entity field.
sortDirection	character(1) direction to sort entities ("asc"ending or "desc"ending) when paging.
filterTerms	character(1) string literal to select rows with an exact (substring) matches in column.
filterOperator	character(1) operator to use when multiple terms in filterTerms=, either "and" (default) or "or".
namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, eg., avworkspace_name().
na	in avtable() and avtable_paged(), character() of strings to be interpretted as missing values. In avtable_import() character(1) value to use for representing NA_character_. See Details.
job_status	tibble() of job identifiers, returned by avtable_import() and avtable_import_set().
path	For avfiles_ls(), the character(1) file or directory path to list. For avfiles_rm(), the character() (perhaps with length greater than 1) of files or directory pat
full_names	logical(1) return names relative to path (FALSE, default) or root of the workspace bucket?
recursive	logical(1) list files recursively?
source	character() file paths. for avfiles_backup(), source can include directory names when recursive = TRUE.

destination	character(1) a google bucket (gs://<bucket-id>/...) to write files. The default is the workspace bucket.
parallel	logical(1) backup files using parallel transfer? See ?avcopy().
project	character(1) project (billing account) name, as returned by, e.g., gcloud_project() or avworkspace_namespace().
account	character(1) google account (email address associated with billing account), as returned by gcloud_account().

Details

avfiles_backup() can be used to back-up individual files or entire directories, recursively. When recursive = FALSE, files are backed up to the bucket with names approximately paste0(destination, "/", basename(source)). When recursive = TRUE and source is a directory path/to/foo/', files are backed up to b "/", dir(basename(source), full.names = TRUE)). Naming conventions are described in detail in gsutil_help("cp")‘.

avfiles_restore() behaves in a manner analogous to avfiles_backup(), copying files from the workspace bucket to the compute node file system.

Value

avtable_paged(): a tibble of data corresponding to the AnVIL table table in the specified workspace.

avfiles_ls() returns a character vector of files in the workspace bucket.

avfiles_backup() returns, invisibly, the status code of the avcopy() command used to back up the files.

avfiles_rm() on success, returns a list of the return codes of avremove(), invisibly.

avruntimes() returns a tibble with columns

- id: integer() runtime identifier.
- googleProject: character() billing account.
- tool: character() e.g., "Jupyter", "RStudio".
- status character() e.g., "Stopped", "Running".
- creator character() AnVIL account, typically "user@gmail.com".
- createdDate character() creation date.
- destroyedDate character() destruction date, or NA.
- dateAccessed character() date of (first?) access.
- runtimeName character().
- clusterServiceAccount character() service ('pet') account for this runtime.
- masterMachineType character() It is unclear which 'tool' populates which of the machineType columns).
- workerMachineType character().
- machineType character().
- persistentDiskId integer() identifier of persistent disk (see avdisks()), or NA.

`avruntime()` returns a tibble witht he same structure as the return value of `avruntimes()`.

`avdisks()` returns a tibble with columns

- id character() disk identifier.
- googleProject: character() billing account.
- status, e.g, "Ready"
- size integer() in GB.
- diskType character().
- blockSize integer().
- creator character() AnVIL account, typically "user@gmail.com".
- createdDate character() creation date.
- destroyedDate character() destruction date, or NA.
- dateAccessed character() date of (first?) access.
- zone character() e.g.. "us-central1-a".
- name character().

Examples

```
library(AnVILBase)
if (has_avworkspace(platform = gcp()) && interactive())
    avfiles_ls()

library(AnVILBase)
if (has_avworkspace(platform = gcp()) && interactive()) {
    ## backup all files in the current directory
    ## default buckets are gs://<bucket-id>/<file-names>
    avfiles_backup(dir())
    ## backup working directory, recursively
    ## default buckets are gs://<bucket-id>/<basename(getwd())>/...
    avfiles_backup(getwd(), recursive = TRUE)
}

if (has_avworkspace(platform = gcp()))
    ## from within AnVIL
    avruntimes()

if (has_avworkspace(strict = TRUE, platform = gcp()))
    ## from within AnVIL
    avdisks()
```

avdata *Import data into the current workspace*

Description

avdata() returns key-value tables representing the information visualized under the DATA tab, 'REFERENCE DATA' and 'OTHER DATA' items. avdata_import() updates (modifies or creates new, but does not delete) rows in 'REFERENCE DATA' or 'OTHER DATA' tables.

Usage

```
avdata(namespace = avworkspace_namespace(), name = avworkspace_name())

avdata_import(
  .data,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)
```

Arguments

namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, eg., avworkspace_name().
.data	A tibble or data.frame for import as an AnVIL table.

Value

avdata() returns a tibble with five columns: "type" represents the origin of the data from the 'REFERENCE' or 'OTHER' data menus. "table" is the table name in the REFERENCE menu, or 'workspace' for the table in the 'OTHER' menu, the key used to access the data element, the value label associated with the data element and the value (e.g., google bucket) of the element.

avdata_import() returns, invisibly, the subset of the input table used to update the AnVIL tables.

Examples

```
library(AnVILBase)
if (has_avworkspace(strict = TRUE, platform = gcp())) {
    ## from within AnVIL
    data <- avdata()
    data
    if (interactive())
        avdata_import(data)
}
```

avnotebooks-methods *Notebook management*

Description

avnotebooks() returns the names of the notebooks associated with the current workspace.

Usage

```
## S4 method for signature 'gcp'
avnotebooks(
  local = FALSE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avnotebooks_localize(
  destination,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avnotebooks_delocalize(
  source,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE,
  ...,
  platform = cloud_platform()
)
```

Arguments

local	= logical(1) notebooks located on the workspace (local = FALSE, default) or runtime / local instance (local = TRUE). When local = TRUE, the notebook path is <avworkspace_name>/notebooks.
namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, eg., avworkspace_name().
...	Additional arguments passed to lower level functions (not used).

platform	gcp() The cloud platform class to dispatch on as given by AnVILBase::cloud_platform. Typically not set manually as cloud_platform() returns the "gcp" class for Google Cloud Platform workspaces on AnVIL.
destination	missing or character(1) file path to the local file system directory for synchronization. The default location is ~/<avworkspace_name>/notebooks. Out-of-date local files are replaced with the workspace version.
dry	logical(1), when TRUE (default), return the consequences of the operation without actually performing the operation.
source	missing or character(1) file path to the local file system directory for synchronization. The default location is ~/<avworkspace_name>/notebooks. Out-of-date local files are replaced with the workspace version.

Value

avnotebooks() returns a character vector of buckets / files located in the workspace 'Files/notebooks' bucket path, or on the local file system.

avnotebooks_localize() returns the exit status of gsutil_rsync().

avnotebooks_delocalize() returns the exit status of gsutil_rsync().

Functions

- avnotebooks(gcp): List notebooks in the workspace
- avnotebooks_localize(gcp): Synchronizes the content of the workspace bucket to the local file system.
- avnotebooks_delocalize(gcp): Synchronizes the content of the notebook location of the local file system to the workspace bucket.

Examples

```
library(AnVILBase)
if (has_avworkspace(strict = TRUE, platform = gcp())) {
    avnotebooks()
    avnotebooks_localize()  # dry run
    try(avnotebooks_delocalize())  # dry run, fails if no local resource
}
```

---

avtable-methods *Methods that work with the primary datasets in the DATA tab*

Description

Tables can be visualized under the DATA tab, TABLES item. avtable() returns an AnVIL table. avtable_paged() retrieves an AnVIL table by requesting the table in 'chunks', and may be appropriate for large tables. avtable_import() imports a data.frame to an AnVIL table. avtable_import_set() imports set membership (i.e., a subset of an existing table) information to an AnVIL table. avtable_delete_values() removes rows from an AnVIL table.

Usage

```
## S4 method for signature 'gcp'
avtables(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avtable(
  table,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  na = c("", "NA"),
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avtable_import(
  .data,
  entity = names(.data)[[1L]],
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  delete_empty_values = FALSE,
  na = "NA",
  n = Inf,
  page = 1L,
  pageSize = NULL,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avtable_import_set(
  .data,
  origin,
  set = names(.data)[[1]],
  member = names(.data)[[2]],
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  delete_empty_values = FALSE,
  na = "NA",
  n = Inf,
  page = 1L,
  pageSize = NULL,
  ...,
```

```
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avtable_delete(
  table,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avtable_delete_values(
  table,
  values,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  ...,
  platform = cloud_platform()
)
```

Arguments

namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, eg., avworkspace_name().
...	Additional arguments passed to lower level functions (not used).
platform	gcp() The cloud platform class to dispatch on as given by AnVILBase::cloud_platform. Typically not set manually as cloud_platform() returns the "gcp" class for Google Cloud Platform workspaces on AnVIL.
table	character(1) table name as returned by, e.g., avtables().
na	in avtable() and avtable_paged(), character() of strings to be interpretted as missing values. In avtable_import() character(1) value to use for representing NA_character_. See Details.
.data	A tibble or data.frame for import as an AnVIL table.
entity	character(1) column name of .data to be used as imported table name. When the table comes from R, this is usually a column name such as sample. The data will be imported into AnVIL as a table sample, with the sample column included with suffix _id, e.g., sample_id. A column in .data with suffix _id can also be used, e.g., entity = "sample_id", creating the table sample with column sample_id in AnVIL. Finally, a value of entity that is not a column in .data, e.g., entity = "unknown", will cause a new table with name entity and entity values seq_len(nrow(.data)).
delete_empty_values	logical(1) when TRUE, remove entities not include in .data from the DATA table. Default: FALSE.

n	numeric(1) maximum number of rows to return
page	integer(1) first page of iteration
pageSize	integer(1) number of records per page. Generally, larger page sizes are more efficient.
origin	character(1) name of the entity (table) used to create the set e.g "sample", "participant", etc.
set	character(1) column name of .data identifying the set(s) to be created.
member	character() vector of entity from the avtable identified by origin. The values may repeat if an ID is in more than one set
values	vector of values in the entity (key) column of table to be deleted. A table sample has an associated entity column with suffix _id, e.g., sample_id. Rows with entity column entries matching values are deleted.

Details

Treatment of missing values in avtable(), avtable_paged() and avtable_import() are handled by the na parameter.

avtable() may sometimes result in a curl error 'Error in curl::curl_fetch_memory' or a 'Internal Server Error (HTTP 500)' This may be due to a server time-out when trying to read a large (more than 50,000 rows?) table; using avtable_paged() may address this problem.

For avtable() and avtable_paged(), the default na = c("", "NA") treats empty cells or cells containing "NA" in a Terra data table as NA_character_ in R. Use na = character() to indicate no missing values, na = "NA" to retain the distinction between "" and NA_character_.

For avtable_import(), the default na = "NA" records NA_character_ in R as the character string "NA" in an AnVIL data table.

The default setting (na = "NA" in avtable_import(), na = c("", NA_character_") in avtable(), is appropriate to 'round-trip' data from R to AnVIL and back when character vectors contain only NA_character_. Use na = "NA" in both functions to round-trip data containing both NA_character_ and "NA". Use a distinct string, e.g., na = "__MISSING_VALUE__", for both arguments if the data contains a string "NA" as well as NA_character_.

avtable_import() tries to work around limitations in .data size in the AnVIL platform, using pageSize (number of rows) to import so that approximately 1500000 elements (rows x columns) are uploaded per chunk. For large .data, a progress bar summarizes progress on the import. Individual chunks may nonetheless fail to upload, with common reasons being an internal server error (HTTP error code 500) or transient authorization failure (HTTP 401). In these and other cases avtable_import() reports the failed page(s) as warnings. The user can attempt to import these individually using the page argument. If many pages fail to import, a strategy might be to provide an explicit pageSize less than the automatically determined size.

avtable_import_set() creates new rows in a table <origin>_set. One row will be created for each distinct value in the column identified by set. Each row entry has a corresponding column <origin> linking to one or more rows in the <origin> table, as given in the member column. The operation is somewhat like split(member, set).

Value

`avtables()`: A tibble with columns identifying the table, the number of records, and the column names.

`avtable()`: a tibble of data corresponding to the AnVIL table `table` in the specified workspace.

`avtable_import_set()` returns a `character(1)` name of the imported AnVIL tibble.

`avtable_delete()` returns `TRUE` if the table is successfully deleted.

`avtable_delete_values()` returns a `tibble` representing deleted entities, invisibly.

Functions

- `avtables(gcp)`: `avtables()` describes tables available in a workspace
- `avtable(gcp)`: `avtable()` retrieves a table from an AnVIL workspace
- `avtable_import(gcp)`: upload a table to the DATA tab
- `avtable_import_set(gcp)`:
- `avtable_delete(gcp)`: Delete a table from the AnVIL workspace.
- `avtable_delete_values(gcp)`:

Examples

```
if (interactive()) {
    avtables("waldronlab-terra", "Tumor_Only_CNV")
    avtable("participant", "waldronlab-terra", "Tumor_Only_CNV")

    library(dplyr)
    ## mtcars dataset
    mtcars_tbl <-
        mtcars |>
        as_tibble(rownames = "model_id") |>
        mutate(model_id = gsub(" ", "-", model_id))

    avworkspace("waldronlab-terra/mramos-wlab-gcp-0")

    avstatus <- avtable_import(mtcars_tbl)

    avtable_import_status(avstatus)

    set_status <- avtable("model") |>
        avtable_import_set("model", "cyl", "model_id")

    avtable_import_status(set_status)

    ## won't be able to delete a row that is referenced in another table
    avtable_delete_values("model", "Mazda-RX4")

    ## delete the set
    avtable_delete("model_set")

    ## then delete the row
```

```
    avtable_delete_values("model", "Mazda-RX4")

    ## recreate the set (if needed)
    avtable("model") |>
        avtable_import_set("model", "cyl", "model_id")

}
library(AnVILBase)
if (has_avworkspace(platform = gcp()) && interactive()) {
## editable copy of '1000G-high-coverage-2019' workspace
avworkspace("bioconductor-rpci-anvil/1000G-high-coverage-2019")
sample <-
    avtable("sample") %>%                               # existing table
    mutate(set = sample(head(LETTERS), nrow(.), TRUE))  # arbitrary groups
sample %>%                                  # new 'participant_set' table
    avtable_import_set("participant", "set", "participant")
sample %>%                                  # new 'sample_set' table
    avtable_import_set("sample", "set", "name")
}
```

`avworkflow-methods` *AnVIL workflow methods*

Description

Methods for working with AnVIL workflow execution. `avworkflow_jobs()` returns a tibble summarizing submitted workflow jobs for a namespace and name.

Usage

```
## S4 method for signature 'gcp'
avworkflow_jobs(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  ...,
  platform = cloud_platform()
)
```

Arguments

`namespace`	`character(1)` AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`
`name`	`character(1)` AnVIL workspace name as returned by, eg., `avworkspace_name()`.
`...`	Additional arguments passed to lower level functions (not used).
`platform`	`gcp()` The cloud platform class to dispatch on as given by AnVILBase::cloud_platform. Typically not set manually as `cloud_platform()` returns the `"gcp"` class for Google Cloud Platform workspaces on AnVIL.

Value

avworkflow_jobs() returns a tibble, sorted by submissionDate, with columns

- submissionId character() job identifier from the workflow runner.
- submitter character() AnVIL user id of individual submitting the job.
- submissionDate POSIXct() date (in local time zone) of job submission.
- status character() job status, with values 'Accepted' 'Evaluating' 'Submitting' 'Submitted' 'Aborting' 'Aborted' 'Done'
- succeeded integer() number of workflows succeeding.
- failed integer() number of workflows failing.

Functions

- avworkflow_jobs(gcp): List workflow jobs in the workspace

Examples

```
library(AnVILBase)
if (has_avworkspace(strict = TRUE, platform = gcp()))
    ## from within AnVIL
    avworkflow_jobs()
```

---

avworkflows *Workflow submissions and file outputs*

Description

avworkflows() returns a tibble summarizing available workflows.

avworkflow_files() returns a tibble containing information and file paths to workflow outputs.

avworkflow_localize() creates or synchronizes a local copy of files with files stored in the workspace bucket and produced by the workflow.

avworkflow_run() runs the workflow of the configuration.

avworkflow_stop() stops the most recently submitted workflow jub from running.

avworkflow_info() returns a tibble containing workflow information, including workflowName, status, start and end time, inputs and outputs.

Usage

```
avworkflows(namespace = avworkspace_namespace(), name = avworkspace_name())

avworkflow_files(
  submissionId = NULL,
  workflowId = NULL,
  bucket,
```

```
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avworkflow_localize(
  submissionId = NULL,
  workflowId = NULL,
  destination = NULL,
  type = c("control", "output", "all"),
  bucket = avstorage(),
  dry = TRUE
)

avworkflow_run(
  config,
  entityName,
  entityType = config$rootEntityType,
  deleteIntermediateOutputFiles = FALSE,
  useCallCache = TRUE,
  useReferenceDisks = FALSE,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)

avworkflow_stop(
  submissionId = NULL,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)

avworkflow_info(
  submissionId = NULL,
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)
```

Arguments

namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, e.g., avworkspace_name().
submissionId	a character() of workflow submission ids, or a tibble with column submissionId, or NULL / missing. See 'Details'.
workflowId	a character(1) of internal identifier associated with one workflow in the submission, or NULL / missing.
bucket	character(1) DEFUNCT - name of the google bucket in which the workflow products are available, as gs://.... Usually the bucket of the active workspace,

returned by avstorage().

destination — character(1) file path to the location where files will be synchronized. For directories in the current working directory, be sure to prepend with "./". When NULL, the submissionId is used as the destination. destination may also be a google bucket, in which case th workflow files are synchronized from the workspace to a second bucket.

type — character(1) copy "control" (default), "output", or "all" files produced by a workflow.

dry — logical(1) when TRUE (default), report the consequences but do not perform the action requested. When FALSE, perform the action.

config — a avworkflow_configuration object of the workflow that will be run. Only entityType and method configuration name and namespace are used from config; other configuration values must be communicated to AnVIL using avworkflow_configuration_set().

entityName — character(1) or NULL name of the set of samples to be used when running the workflow. NULL indicates that no sample set will be used.

entityType — character(1) or NULL type of root entity used for the workflow. NULL means that no root entity will be used.

deleteIntermediateOutputFiles — logical(1) whether or not to delete intermediate output files when the workflow completes.

useCallCache — logical(1) whether or not to read from cache for this submission.

useReferenceDisks — logical(1) whether or not to use pre-built disks for common genome references. Default: FALSE.

Details

For avworkflow_files(), the submissionId is the identifier associated with the submission of one (or more) workflows, and is present in the return value of avworkflow_jobs(); the example illustrates how the first row of avworkflow_jobs() (i.e., the most recently completed workflow) can be used as input to avworkflow_files(). When submissionId is not provided, the return value is for the most recently submitted workflow of the namespace and name of avworkspace().

avworkflow_localize(). type = "control" files summarize workflow progress; they can be numerous but are frequently small and quickly syncronized. type = "output" files are the output products of the workflow stored in the workspace bucket. Depending on the workflow, outputs may be large, e.g., aligned reads in bam files. See avcopy() to copy individual files from the bucket to the local drive.

avworkflow_localize() treats submissionId= in the same way as avworkflow_files(): when missing, files from the most recent workflow job are candidates for localization.

Value

avworkflows() returns a tibble. Each workflow is in a 'namespace' and has a 'name', as illustrated in the example. Columns are

- name: workflow name.

- namespace: workflow namespace (often the same as the workspace namespace).
- rootEntityType: name of the `avtable()` used to retrieve inputs.
- methodRepoMethod.methodUri: source of the method, e.g., a dockstore URI.
- methodRepoMethod.sourceRepo: source repository, e.g., dockstore.
- methodRepoMethod.methodPath: path to method, e.g., a dockerstore method might reference a github repository.
- methodRepoMethod.methodVersion: the version of the method, e.g., 'main' branch of a github repository.

`avworkflow_files()` returns a tibble with columns

- file: character() 'base name' of the file in the bucket.
- workflow: character() name of the workflow the file is associated with.
- task: character() name of the task in the workflow that generated the file.
- path: charcter() full path to the file in the google bucket.
- submissionId: character() internal identifier associated with the submission the files belong to.
- workflowId: character() internal identifer associated with each workflow (e.g., row of an avtable() used as input) in the submission.
- submissionRoot: character() path in the workspace bucket to the root of files created by this submission.
- namespace: character() AnVIL workspace namespace (billing account) associated with the submissionId.
- name: character(1) AnVIL workspace name associated with the submissionId.

`avworkflow_localize()` prints a message indicating the number of files that are (if `dry = FALSE`) or would be localized. If no files require localization (i.e., local files are not older than the bucket files), then no files are localized. `avworkflow_localize()` returns a tibble of file name and bucket path of files to be synchronized.

`avworkflow_run()` returns `config`, invisibly.

`avworkflow_stop()` returns (invisibly) `TRUE` on successfully requesting that the workflow stop, `FALSE` if the workflow is already aborting, aborted, or done.

`avworkflow_info()` returns a tibble with columns: submissionId, workflowId, workflowName,status, start, end, inputs and outputs.

Examples

```
library(AnVILBase)
if (has_avworkspace(strict = TRUE, platform = gcp()))
    ## from within AnVIL
    avworkflows() %>% select(namespace, name)

if (has_avworkspace(strict = TRUE, platform = gcp())) {
    ## e.g., from within AnVIL
    jobs <- avworkflow_jobs()
    if (nrow(jobs)) {
        jobs |>
```

```
        ## select most recent workflow
        head(1) |>
        ## find paths to output and log files on the bucket
        avworkflow_files()
    }
}

if (has_avworkspace(strict = TRUE, platform = gcp()))
    avworkflow_localize(dry = TRUE)

if (has_avworkspace(strict = TRUE, platform = gcp()) && interactive()) {
    entityName <- avtable("participant_set") |>
        pull(participant_set_id) |>
        head(1)
    avworkflow_run(new_config, entityName)
}

if (has_avworkspace(strict = TRUE, platform = gcp()) && interactive()) {
    avworkflow_stop()
}
if (has_avworkspace(strict = TRUE, platform = gcp()))
    avworkflow_info()
```

`avworkflow_configurations` *Workflow configuration*

Description

Funtions on this help page facilitate getting, updating, and setting workflow configuration parameters. See `?avworkflow` for additional relevant functionality.

`avworkflow_namespace()` and `avworkflow_name()` are utility functions to record the workflow namespace and name required when working with workflow configurations. `avworkflow()` provides a convenient way to provide workflow namespace and name in a single command, `namespace/name`.

`avworkflow_configuration_get()` returns a list structure describing an existing workflow configuration.

`avworkflow_configuration_inputs()` returns a data.frame template for the inputs defined in a workflow configuration. This template can be used to provide custom inputs for a configuration.

`avworkflow_configuration_outputs()` returns a data.frame template for the outputs defined in a workflow configuration. This template can be used to provide custom outputs for a configuration.

`avworkflow_configuration_update()` returns a list structure describing a workflow configuration with updated inputs and / or outputs.

`avworkflow_configuration_set()` updates an existing configuration in Terra / AnVIL, e.g., changing inputs to the workflow.

`avworkflow_configuration_template()` returns a template for defining workflow configurations. This template can be used as a starting point for providing a custom configuration.

Usage

```
avworkflow_namespace(workflow_namespace = NULL)

avworkflow_name(workflow_name = NULL)

avworkflow(workflow = NULL)

avworkflow_configuration_get(
  workflow_namespace = avworkflow_namespace(),
  workflow_name = avworkflow_name(),
  namespace = avworkspace_namespace(),
  name = avworkspace_name()
)

avworkflow_configuration_inputs(config)

avworkflow_configuration_outputs(config)

avworkflow_configuration_update(
  config,
  inputs = avworkflow_configuration_inputs(config),
  outputs = avworkflow_configuration_outputs(config)
)

avworkflow_configuration_set(
  config,
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  dry = TRUE
)

avworkflow_configuration_template()

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

Arguments

workflow_namespace	character(1) AnVIL workflow namespace, as returned by, e.g., the namespace column of avworkflows().
workflow_name	character(1) AnVIL workflow name, as returned by, e.g., the name column of avworkflows().
workflow	character(1) representing the combined workflow namespace and name, as namespace/name.
namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, e.g., avworkspace_name().

config	a named list describing the full configuration, e.g., created from editing the return value of avworkflow_configuration_set() or avworkflow_configuration_template().
inputs	the new inputs to be updated in the workflow configuration. If none are specified, the inputs from the original configuration will be used and no changes will be made.
outputs	the new outputs to be updated in the workflow configuration. If none are specified, the outputs from the original configuration will be used and no changes will be made.
dry	logical(1) when TRUE (default), report the consequences but do not perform the action requested. When FALSE, perform the action.
x	Object of class avworkflow_configuration.
...	additional arguments to print(); unused.

Details

The exact format of the configuration is important.

One common problem is that a scalar character vector "bar" is interpretted as a json 'array' ["bar"] rather than a json string "bar". Enclose the string with jsonlite::unbox("bar") in the configuration list if the length 1 character vector in R is to be interpretted as a json string.

A second problem is that an unquoted unboxed character string unbox("foo") is required by AnVIL to be quoted. This is reported as a warning() about invalid inputs or outputs, and the solution is to provide a quoted string unbox('"foo"').

Value

avworkflow_namespace(), and avworkflow_name() return character(1) identifiers. avworkflow() returns the character(1) concatenated namespace and name. The value returned by avworkflow_name() will be percent-encoded (e.g., spaces " " replaced by "%20").

avworkflow_configuration_get() returns a list structure describing the configuration. See avworkflow_configuration_ for the structure of a typical workflow.

avworkflow_configuration_inputs() returns a data.frame providing a template for the configuration inputs, with the following columns:

- inputType
- name
- optional
- attribute

The only column of interest to the user is the attribute column, this is the column that should be changed for customization.

avworkflow_configuration_outputs() returns a data.frame providing a template for the configuration outputs, with the following columns:

- name
- outputType

- attribute

The only column of interest to the user is the `attribute` column, this is the column that should be changed for customization.

`avworkflow_configuration_update()` returns a list structure describing the updated configuration.

`avworkflow_configuration_set()` returns an object describing the updated configuration. The return value includes invalid or unused elements of the `config` input. Invalid or unused elements of `config` are also reported as a warning.

`avworkflow_configuration_template()` returns a list providing a template for configuration lists, with the following structure:

- namespace character(1) configuration namespace.
- name character(1) configuration name.
- rootEntityType character(1) or missing. the name of the table (from `avtables()`) containing the entitites referenced in inputs, etc., by the keyword 'this.'
- prerequisites named list (possibly empty) of prerequisites.
- inputs named list (possibly empty) of inputs. Form of input depends on method, and might include, e.g., a reference to a field in a table referenced by `avtables()` or a character string defining an input constant.
- outputs named list (possibly empty) of outputs.
- methodConfigVersion integer(1) identifier for the method configuration.
- methodRepoMethod named list describing the method, with character(1) elements described in the return value for `avworkflows()`.
  - methodUri
  - sourceRepo
  - methodPath
  - methodVersion. The REST specification indicates that this has type `integer`, but the documentation indicates either `integer` or `string`.
- deleted logical(1) of uncertain purpose.

See Also

The help page `?avworkflow` for discovering, running, stopping, and retrieving outputs from workflows.

Examples

```
if (has_avworkspace(platform = gcp()) && interactive()) {
    ## set the namespace and name as appropriate
    avworkspace("bioconductor-rpci-anvil/Bioconductor-Workflow-DESeq2")

    ## discover available workflows in the workspace
    avworkflows()

    ## record the workflow of interest
```

```
    avworkflow("bioconductor-rpci-anvil/AnVILBulkRNASeq")

    ## what workflows are available?
    available_workflows <- avworkflows()

    ## retrieve the current configuration
    config <- avworkflow_configuration_get()
    config

    ## what are the inputs and outputs?
    inputs <- avworkflow_configuration_inputs(config)
    inputs

    outputs <- avworkflow_configuration_outputs(config)
    outputs

    ## update inputs or outputs, e.g., this input can be anything...
    inputs <-
        inputs |>
        dplyr::mutate(attribute = ifelse(
            name == "salmon.transcriptome_index_name",
            '"new_index_name"',
            attribute
        ))
    new_config <- avworkflow_configuration_update(config, inputs)
    new_config

    ## set the new configuration in AnVIL; use dry = FALSE to actually
    ## update the configuration
    avworkflow_configuration_set(config)
}

## avworkflow_configuration_template() is a utility function that may
## help understanding what the inputs and outputs should be
avworkflow_configuration_template() |>
    str()

avworkflow_configuration_template()
```

`avworkspace-methods` *AnVIL Workspace GCP methods*

Description

`avworkspace_namespace()` and `avworkspace_name()` are utiliity functions to retrieve workspace namespace and name from environment variables or interfaces usually available in AnVIL notebooks or RStudio sessions. `avworkspace()` provides a convenient way to specify workspace namespace and name in a single command. `avworkspace_clone()` clones (copies) an existing workspace, possibly into a new namespace (billing account).

Usage

```
## S4 method for signature 'gcp'
avworkspaces(..., platform = cloud_platform())

## S4 method for signature 'gcp'
avworkspace_namespace(
  namespace = NULL,
  warn = TRUE,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avworkspace_name(name = NULL, warn = TRUE, ..., platform = cloud_platform())

## S4 method for signature 'gcp'
avworkspace(workspace = NULL, ..., platform = cloud_platform())

## S4 method for signature 'gcp'
avworkspace_clone(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  to_namespace = namespace,
  to_name,
  storage_region = "US",
  bucket_location,
  ...,
  platform = cloud_platform()
)
```

Arguments

`...`	additional arguments passed as-is to the `gsutil` subcommand.
`platform`	`gcp()` The cloud platform class to dispatch on as given by AnVILBase::cloud_platform. Typically not set manually as `cloud_platform()` returns the `"gcp"` class for Google Cloud Platform workspaces on AnVIL.
`namespace`	`character(1)` AnVIL workspace namespace as returned by, e.g., `avworkspace_namespace()`
`warn`	logical(1) when `TRUE` (default), generate a warning when the workspace namespace or name cannot be determined.
`name`	`character(1)` AnVIL workspace name as returned by, e.g., `avworkspace_name()`.
`workspace`	when present, a `character(1)` providing the concatenated namespace and name, e.g., `"bioconductor-rpci-anvil/Bioconductor-Package-AnVIL"`
`to_namespace`	character(1) workspace (billing account) in which to make the clone.
`to_name`	character(1) name of the cloned workspace.
`storage_region`	character(1) region (NO multi-region, except the default) in which bucket attached to the workspace should be created.

bucket_location
: character(1) DEFUNCT; use storage_region instead. Region (NO multi-region, except the default) in which bucket attached to the workspace should be created.

Details

avworkspace_namespace() is the billing account. If the namespace= argument is not provided, try gcloud_project(), and if that fails try Sys.getenv("WORKSPACE_NAMESPACE").

avworkspace_name() is the name of the workspace as it appears in https://app.terra.bio/#workspaces. If not provided, avworkspace_name() tries to use Sys.getenv("WORKSPACE_NAME").

Namespace and name values are cached across sessions, so explicitly providing avworkspace_name*() is required at most once per session. Revert to system settings with arguments NA.

Value

avworkspace_namespace(), and avworkspace_name() return character(1) identifiers. avworkspace() returns the character(1) concatenated namespace and name. The value returned by avworkspace_name() will be percent-encoded (e.g., spaces " " replaced by "%20").

avworkspace_clone() returns the namespace and name, in the format namespace/name, of the cloned workspace.

Functions

- avworkspaces(gcp): list workspaces in the current project as a tibble
- avworkspace_namespace(gcp): Get or set the namespace of the current workspace
- avworkspace_name(gcp): Get or set the name of the current workspace
- avworkspace(gcp): Get the current workspace namespace and name combination
- avworkspace_clone(gcp): Clone the current workspace

Examples

```
if (has_avworkspace(platform = gcp())) {
    avworkspaces()
    avworkspace_namespace()
    avworkspace_name()
    avworkspace()
}
```

---

drs *DRS (Data Repository Service) URL management*

Description

drs_hub() resolves zero or more DRS URLs to their Google bucket location using the DRS Hub API endpoint.

Usage

```
drs_hub(source = character())

drs_nci_crdc(source = character())
```

Arguments

source	character() DRS URLs (beginning with 'drs://') to resources managed by the DRS Hub server (drs_hub()).

Value

drs_hub() returns a tbl with the following columns:

- drs: character() DRS URIs
- bucket: character() Google cloud bucket
- name: character() object name in bucket
- size: numeric() object size in bytes
- timeCreated: character() object creation time
- timeUpdated: character() object update time
- fileName: character() local file name
- accessUrl: character() signed URL for object access

drs_hub

drs_hub() uses the DRS Hub API endpoint to resolve a single or multiple DRS URLs to their Google bucket location. The DRS Hub API endpoint requires a gcloud_access_token(). The DRS Hub API service is hosted at https://drshub.dsde-prod.broadinstitute.org.

drs_nci_crdc

drs_nci_crdc() resolves one or more DRS URLs to their <gdc.cancer.gov> location. The implementation allows the extraction of access_url values to download the DRS objects. The DRS NCI CRDC service is hosted at https://nci-crdc.datacommons.io.

Examples

```
if (GCPtools::gcloud_exists() && interactive()) {
    drs_urls <- c(
        "drs://drs.anv0:v2_b3b815c7-b012-37b8-9866-1cb44b597924",
        "drs://drs.anv0:v2_2823eac3-77ae-35e4-b674-13dfab629dc5",
        "drs://drs.anv0:v2_c6077800-4562-30e3-a0ff-aa03a7e0e24f"
    )
    drs_hub(drs_urls)

    drs_nci <- c(
        "drs://nci-crdc.datacommons.io/56e35487-b20f-45ba-8d84-9f16b26c85ea",
        "drs://nci-crdc.datacommons.io/f814f1ec-6850-4ab6-ac0f-df9f77ee185b",
```

```
        "drs://nci-crdc.datacommons.io/d9b591d5-7fe8-43fe-b0b3-4fc0f9736866"
    )
    drs_nci_crdc(drs_nci)
}
```

---

gcloud-defunct *gcloud command line utility interface (DEFUNCT)*

Description

These functions invoke the gcloud command line utility. See GCPtools::gsutil for details on how gcloud is located. **NOTE**. These functions have been moved to the GCPtools package.

gcloud_access_token() generates a token for the given service account. The token is cached for the duration of its validity. The token is refreshed when it expires. The token is obtained using the gcloud command line utility for the given gcloud_account(). The function is mainly used internally by API service functions, e.g., AnVIL::Terra()

gcloud_exists() tests whether the gcloud() command can be found on this system. After finding the binary location, it runs gcloud version to identify potentially misconfigured installations. See 'Details' section of gsutil for where the application is searched.

gcloud_account(): report the current gcloud account via gcloud config get-value account.

gcloud_project(): report the current gcloud project via gcloud config get-value project.

gcloud_help(): queries gcloud for help for a command or sub-comand via gcloud help ....

gcloud_cmd() allows arbitrary gcloud command execution via gcloud .... Use pre-defined functions in preference to this.

gcloud_storage() allows arbitrary gcloud storage command execution via gcloud storage .... Typically used for bucket management commands such as rm and cp.

gcloud_storage_buckets() provides an interface to the gcloud storage buckets command. This command can be used to create a new bucket via gcloud storage buckets create ....

Usage

```
gcloud_access_token(service)

gcloud_exists()

gcloud_account(account = NULL)

gcloud_project(project = NULL)

gcloud_help(...)

gcloud_cmd(cmd, ...)

gcloud_storage(cmd, ...)

gcloud_storage_buckets(bucket_cmd = "create", bucket, ...)
```

Arguments

service	character(1) The name of the service, e.g. "terra" for which to obtain an access token for.
account	character(1) Google account (e.g., user@gmail.com) to use for authentication.
project	character(1) billing project name.
...	Additional arguments appended to gcloud commands.
cmd	character(1) representing a command used to evaluate gcloud cmd ....
bucket_cmd	character(1) representing a buckets command typically used to create a new bucket. It can also be used to add-iam-policy-binding or remove-iam-policy-binding to a bucket.
bucket	character(1) representing a unique bucket name to be created or modified.

Value

gcloud_access_token() returns a simple token string to be used with the given service.

gcloud_exists() returns TRUE when the gcloud application can be found, FALSE otherwise.

gcloud_account() returns a character(1) vector containing the active gcloud account, typically a gmail email address.

gcloud_project() returns a character(1) vector containing the active gcloud project.

gcloud_help() returns an unquoted character() vector representing the text of the help manual page returned by gcloud help ....

gcloud_cmd() returns a character() vector representing the text of the output of gcloud cmd ...

---

gcp-class *GCP platform class*

Description

This class is used to represent the GCP platform.

Usage

```
gcp()
```

Value

An object of class gcp.

Examples

```
showClass("gcp")
```

gcp-methods *Methods compatible with the GCP platform class*

Description

`avcopy()`: copy contents of `source` to `destination`. At least one of `source` or `destination` must be Google cloud bucket; `source` can be a character vector with length greater than 1. Use `gsutil_help("cp")` for `gsutil` help.

`avlist()`: List contents of a google cloud bucket or, if `source` is missing, all Cloud Storage buckets under your default project ID

`avremove()`: remove contents of a Google Cloud Bucket.

`avbackup()`,`avrestore()`: synchronize a source and a destination. If the destination is on the local file system, it must be a directory or not yet exist (in which case a directory will be created).

`avstorage()` returns the workspace bucket, i.e., the google bucket associated with a workspace. Bucket content can be visualized under the 'DATA' tab, 'Files' item.

`avworkspaces()`: returns a tibble with columns including the name, last modification time, namespace, and owner status.

`avtable_import()`: returns a `tibble()` containing the page number, 'from' and 'to' rows included in the page, job identifier, initial status of the uploaded 'chunks', and any (error) messages generated during status check. Use `avtable_import_status()` to query current status.

Usage

```
## S4 method for signature 'gcp'
avcopy(
  source,
  destination,
  ...,
  recursive = FALSE,
  parallel = TRUE,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avlist(
  source = character(),
  recursive = FALSE,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avremove(
  source,
  recursive = FALSE,
```

```
  force = FALSE,
  parallel = TRUE,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avbackup(
  source,
  destination,
  recursive = FALSE,
  exclude = NULL,
  dry = TRUE,
  delete = FALSE,
  parallel = TRUE,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avrestore(
  source,
  destination,
  recursive = FALSE,
  exclude = NULL,
  dry = TRUE,
  delete = FALSE,
  parallel = TRUE,
  ...,
  platform = cloud_platform()
)

## S4 method for signature 'gcp'
avstorage(
  namespace = avworkspace_namespace(),
  name = avworkspace_name(),
  ...,
  platform = cloud_platform()
)
```

Arguments

source	character(1), (character() for avlist(), avcopy()) paths to a google storage bucket, possibly with wild-cards for file-level pattern matching.
destination	character(1), google cloud bucket or local file system destination path.
...	additional arguments passed as-is to the gsutil subcommand.
recursive	logical(1); perform operation recursively from source?. Default: FALSE.
parallel	logical(1), perform parallel multi-threaded / multi-processing (default is TRUE).

platform	gcp() The cloud platform class to dispatch on as given by AnVILBase::cloud_platform. Typically not set manually as cloud_platform() returns the "gcp" class for Google Cloud Platform workspaces on AnVIL.
force	logical(1): continue silently despite errors when removing multiple objects. Default: FALSE.
exclude	character(1) a python regular expression of bucket paths to exclude from synchronization. E.g., '.*(\\.png|\\.txt)$" excludes '.png' and .txt' files.
dry	logical(1), when TRUE (default), return the consequences of the operation without actually performing the operation.
delete	logical(1), when TRUE, remove files in destination that are not in source. Exercise caution when you use this option: it's possible to delete large amounts of data accidentally if, for example, you erroneously reverse source and destination.
namespace	character(1) AnVIL workspace namespace as returned by, e.g., avworkspace_namespace()
name	character(1) AnVIL workspace name as returned by, e.g., avworkspace_name().

Details

avbackup()': To make "gs://mybucket/data"match the contents of the local directory"data"‘ you could do:

```
avbackup("data", "gs://mybucket/data", delete = TRUE)
```

To make the local directory "data" the same as the contents of gs://mybucket/data:

```
avrestore("gs://mybucket/data", "data", delete = TRUE)
```

If destination is a local path and does not exist, it will be created.

Value

avcopy(): exit status of avcopy(), invisibly. avlist(): character() listing of source content. avremove(): exit status of gsutil rm, invisibly. avbackup(): exit status of gsutil rsync, invisibly. avrestore(): exit status of gsutil rsync, invisbly. avstorage() returns a character(1) bucket identifier prefixed with gs://

Functions

- avcopy(gcp): copy contents of source to destination with gsutil
- avlist(gcp): list contents of source with gsutil
- avremove(gcp): remove contents of source with gsutil
- avbackup(gcp): backup contents of source with gsutil
- avrestore(gcp): restore contents of source with gsutil
- avstorage(gcp): get the storage bucket location

Examples

```
src <-
   "gs://genomics-public-data/1000-genomes/other/sample_info/sample_info.csv"
if (has_avworkspace(platform = gcp())) {
   avcopy(src, tempdir())
   ## internal gsutil_*() commands work with spaces in source or destination
   destination <- file.path(tempdir(), "foo bar")
   avcopy(src, destination)
   file.exists(destination)
}
if (has_avworkspace(strict = TRUE, platform = gcp()))
    ## From within AnVIL...
    bucket <- avstorage()                       # discover bucket

if (has_avworkspace(strict = TRUE, platform = gcp()) && interactive()) {
    path <- file.path(bucket, "mtcars.tab")
    avlist(dirname(path))                    # no 'mtcars.tab'...
    write.table(mtcars, gsutil_pipe(path, "w")) # write to bucket
    gsutil_stat(path)                           # yep, there!
    read.table(gsutil_pipe(path, "r"))          # read from bucket
}
```

`gsutil-deprecated` *gsutil command line utility interface (DEPRECATED)*

Description

These functions invoke the `gsutil` command line utility. See the "Details:" section if you have gsutil installed but the package cannot find it. These functions have been moved to the `GCPtools` package.

`gsutil_requesterpays()`: does the google bucket require that the requester pay for access?

`gsutil_exists()`: check if the bucket or object exists.

`gsutil_stat()`: print, as a side effect, the status of a bucket, directory, or file.

`gsutil_rsync()`: synchronize a source and a destination. If the destination is on the local file system, it must be a directory or not yet exist (in which case a directory will be created).

`gsutil_cat()`: concatenate bucket objects to standard output

`gsutil_help()`: print 'man' page for the `gsutil` command or subcommand. Note that only commandes documented on this R help page are supported.

`gsutil_pipe()`: create a pipe to read from or write to a gooogle bucket object.

Usage

```
gsutil_requesterpays(source)

gsutil_exists(source)
```

```
gsutil_stat(source)

gsutil_rsync(
  source,
  destination,
  ...,
  exclude = NULL,
  dry = TRUE,
  delete = FALSE,
  recursive = FALSE,
  parallel = TRUE
)

gsutil_cat(source, ..., header = FALSE, range = integer())

gsutil_help(cmd = character(0))

gsutil_pipe(source, open = "r", ...)
```

Arguments

source	character() for gsutil_requesterpays() and gsutil_exists(): paths to a Google Storage Bucket, possibly with wild-cards for file-level pattern matching.
destination	character(1), google cloud bucket or local file system destination path.
...	additional arguments passed as-is to the gsutil subcommand.
exclude	character(1) a python regular expression of bucket paths to exclude from synchronization. E.g., '.*(\\.png|\\.txt)$" excludes '.png' and .txt' files.
dry	logical(1), when TRUE (default), return the consequences of the operation without actually performing the operation.
delete	logical(1), when TRUE, remove files in destination that are not in source. Exercise caution when you use this option: it's possible to delete large amounts of data accidentally if, for example, you erroneously reverse source and destination.
recursive	logical(1); perform operation recursively from source?. Default: FALSE.
parallel	logical(1), perform parallel multi-threaded / multi-processing (default is TRUE).
header	logical(1) when TRUE annotate each
range	(optional) integer(2) vector used to form a range from-to of bytes to concatenate. NA values signify concatenation from the start (first position) or to the end (second position) of the file.
cmd	character() (optional) command name, e.g., "ls" for help.
open	character(1) either "r" (read) or "w" (write) from the bucket.

Details

The gsutil system command is required. The search for gsutil starts with environment variable GCLOUD_SDK_PATH providing a path to a directory containing a bin directory containingin gsutil,

gcloud, etc. The path variable is searched for first as an option() and then system variable. If no option or global variable is found, Sys.which() is tried. If that fails, gsutil is searched for on defined paths. On Windows, the search tries to find Google\\Cloud SDK\\google-cloud-sdk\\bin\\gsutil.cmd in the LOCAL APP DATA, Program Files, and Program Files (x86) directories. On linux / macOS, the search continues with ~/google-cloud-sdk.
gsutil_rsync()': To make "gs://mybucket/data"match the contents of the local directory"data"‘ you could do:

gsutil_rsync("data", "gs://mybucket/data", delete = TRUE)

To make the local directory "data" the same as the contents of gs://mybucket/data:

gsutil_rsync("gs://mybucket/data", "data", delete = TRUE)

If destination is a local path and does not exist, it will be created.

Value

gsutil_requesterpays(): named logical() vector TRUE when requester-pays is enabled.

gsutil_exists(): logical(1) TRUE if bucket or object exists.

gsutil_stat(): tibble() summarizing status of each bucket member.

gsutil_rsync(): exit status of gsutil_rsync(), invisbly.

gsutil_cat() returns the content as a character vector.

gsutil_help(): character() help text for subcommand cmd.

gsutil_pipe() an unopened R pipe(); the mode is *not* specified, and the pipe must be used in the appropriate context (e.g., a pipe created with open = "r" for input as read.csv())

Examples

```
src <-
  "gs://genomics-public-data/1000-genomes/other/sample_info/sample_info.csv"
if (has_avworkspace(platform = gcp()))
    GCPtools::gsutil_requesterpays(src) # FALSE -- no cost download

if (has_avworkspace(platform = gcp())) {
    GCPtools::gsutil_exists(src)
    GCPtools::gsutil_stat(src)
    avlist(dirname(src))
}

if (has_avworkspace(platform = gcp()))
    GCPtools::gsutil_help("ls")

if (has_avworkspace(platform = gcp())) {
    df <- read.csv(gsutil_pipe(src), 5L)
    class(df)
    dim(df)
    head(df)
}
```

has_avworkspace-methods

Helper to check AnVIL environment is set up to work with GCP

Description

`has_avworkspace()` checks that the AnVIL environment is set up to work with GCP. If `strict = TRUE`, it also checks that the workspace name is set.

Usage

```
## S4 method for signature 'gcp'
has_avworkspace(strict = FALSE, ..., platform = cloud_platform())
```

Arguments

`strict`	`logical(1)` Whether to include a check for an existing `avworkspace_name()` setting. Default `FALSE`.
`...`	Arguments passed to the methods.
`platform`	A Platform derived class indicating the AnVIL environment, currently, `azure` and `gcp` classes are compatible.

Value

`logical(1)` `TRUE` if the AnVIL environment is set up properly to interact with GCP, otherwise `FALSE`.

Functions

- `has_avworkspace(gcp)`: Check if the AnVIL environment is set up

Examples

```
has_avworkspace(platform = gcp())
```

localize

Copy packages, folders, or files to or from google buckets.

Description

`localize()`: recursively synchronizes files from a Google storage bucket (`source`) to the local file system (`destination`). This command acts recursively on the `source` directory, and does not delete files in `destination` that are not in 'source.

`delocalize()`: synchronize files from a local file system (`source`) to a Google storage bucket (`destination`). This command acts recursively on the `source` directory, and does not delete files in `destination` that are not in `source`.

Usage

```
localize(source, destination, dry = TRUE)

delocalize(source, destination, unlink = FALSE, dry = TRUE)
```

Arguments

source	character(1), a google storage bucket or local file system directory location.
destination	character(1), a google storage bucket or local file system directory location.
dry	logical(1), when TRUE (default), return the consequences of the operation without actually performing the operation.
unlink	logical(1) remove (unlink) the file or directory in source. Default: FALSE.

Value

localize(): exit status of function gsutil_rsync().

delocalize(): exit status of function gsutil_rsync()

Index

.gcp *(gcp-class)*, 28

AnVILBase::cloud_platform, *9*, *11*, *14*, *24*, *31*
av, 2
avbackup *(gcp-methods)*, 29
avbackup,gcp-method *(gcp-methods)*, 29
avcopy *(gcp-methods)*, 29
avcopy,gcp-method *(gcp-methods)*, 29
avdata, 7
avdata_import *(avdata)*, 7
avdisks *(av)*, 2
avfiles_backup *(av)*, 2
avfiles_ls *(av)*, 2
avfiles_restore *(av)*, 2
avfiles_rm *(av)*, 2
avlist *(gcp-methods)*, 29
avlist,gcp-method *(gcp-methods)*, 29
avnotebooks *(avnotebooks-methods)*, 8
avnotebooks,gcp-method *(avnotebooks-methods)*, 8
avnotebooks-methods, 8
avnotebooks_delocalize *(avnotebooks-methods)*, 8
avnotebooks_delocalize,gcp-method *(avnotebooks-methods)*, 8
avnotebooks_localize *(avnotebooks-methods)*, 8
avnotebooks_localize,gcp-method *(avnotebooks-methods)*, 8
avremove *(gcp-methods)*, 29
avremove,gcp-method *(gcp-methods)*, 29
avrestore *(gcp-methods)*, 29
avrestore,gcp-method *(gcp-methods)*, 29
avruntime *(av)*, 2
avruntimes *(av)*, 2
avstorage *(gcp-methods)*, 29
avstorage,gcp-method *(gcp-methods)*, 29
avtable *(avtable-methods)*, 9
avtable,gcp-method *(avtable-methods)*, 9
avtable-methods, 9
avtable_delete,gcp-method *(avtable-methods)*, 9
avtable_delete_values *(avtable-methods)*, 9
avtable_delete_values,gcp-method *(avtable-methods)*, 9
avtable_import *(avtable-methods)*, 9
avtable_import,gcp-method *(avtable-methods)*, 9
avtable_import_set *(avtable-methods)*, 9
avtable_import_set,gcp-method *(avtable-methods)*, 9
avtable_import_status *(av)*, 2
avtable_paged *(av)*, 2
avtables *(avtable-methods)*, 9
avtables,gcp-method *(avtable-methods)*, 9
avworkflow *(avworkflow_configurations)*, 19
avworkflow-methods, 14
avworkflow_configuration_get *(avworkflow_configurations)*, 19
avworkflow_configuration_inputs *(avworkflow_configurations)*, 19
avworkflow_configuration_outputs *(avworkflow_configurations)*, 19
avworkflow_configuration_set *(avworkflow_configurations)*, 19
avworkflow_configuration_template *(avworkflow_configurations)*, 19
avworkflow_configuration_update *(avworkflow_configurations)*, 19
avworkflow_configurations, 19
avworkflow_files *(avworkflows)*, 15
avworkflow_info *(avworkflows)*, 15
avworkflow_jobs,gcp-method *(avworkflow-methods)*, 14
avworkflow_localize *(avworkflows)*, 15
avworkflow_name

(avworkflow_configurations), 19
avworkflow_namespace
(avworkflow_configurations), 19
avworkflow_run (avworkflows), 15
avworkflow_stop (avworkflows), 15
avworkflows, 15
avworkspace,gcp-method
(avworkspace-methods), 23
avworkspace-methods, 23
avworkspace_clone,gcp-method
(avworkspace-methods), 23
avworkspace_name,gcp-method
(avworkspace-methods), 23
avworkspace_namespace,gcp-method
(avworkspace-methods), 23
avworkspaces,gcp-method
(avworkspace-methods), 23

delocalize (localize), 35
drs, 25
drs_hub (drs), 25
drs_nci_crdc (drs), 25

gcloud-defunct, 27
gcloud_access_token (gcloud-defunct), 27
gcloud_account (gcloud-defunct), 27
gcloud_cmd (gcloud-defunct), 27
gcloud_exists (gcloud-defunct), 27
gcloud_help (gcloud-defunct), 27
gcloud_project (gcloud-defunct), 27
gcloud_storage (gcloud-defunct), 27
gcloud_storage_buckets
(gcloud-defunct), 27
gcp (gcp-class), 28
gcp-class, 28
gcp-methods, 29
GCPtools::gsutil, 27
gsutil-deprecated, 32
gsutil_cat (gsutil-deprecated), 32
gsutil_exists (gsutil-deprecated), 32
gsutil_help (gsutil-deprecated), 32
gsutil_pipe (gsutil-deprecated), 32
gsutil_requesterpays
(gsutil-deprecated), 32
gsutil_rsync (gsutil-deprecated), 32
gsutil_stat (gsutil-deprecated), 32

has_avworkspace,gcp-method
(has_avworkspace-methods), 35
has_avworkspace-methods, 35

localize, 35

print.avworkflow_configuration
(avworkflow_configurations), 19