Package 'pins'

January 21, 2023

Type Package

Title Pin, Discover and Share Resources

Version 1.1.0

Description Publish data sets, models, and other R objects, making it
easy to share them across projects and with your colleagues. You can
pin objects to a variety of ``boards'', including local folders (to
share on a networked drive or with 'DropBox'), 'RStudio' connect,
Amazon S3, and more.

License Apache License (>= 2)

URL <https://pins.rstudio.com/>, <https://github.com/rstudio/pins-r>

BugReports <https://github.com/rstudio/pins-r/issues>

Depends R (>= 3.3.0)

Imports cli, digest, ellipsis, filelock, fs, generics, glue, httr,
jsonlite, lifecycle, magrittr, mime, openssl, rappdirs, rlang
(>= 0.4.10), tibble, whisker, withr (>= 2.4.3), yaml, zip

Suggests archive, arrow, AzureStor, data.table, datasets, gitcreds,
googleCloudStorageR, knitr, Microsoft365R, mockery,
paws.storage, qs, R.utils, rmarkdown, rsconnect, shiny, sodium,
testthat (>= 3.0.0), webfakes, xml2

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Julia Silge [cre, aut] (<<https://orcid.org/0000-0002-3671-836X>>),
Hadley Wickham [aut] (<<https://orcid.org/0000-0003-4757-117X>>),
Javier Luraschi [aut],
RStudio [cph, fnd]

Maintainer Julia Silge <julia.silge@rstudio.com>

Repository CRAN

Date/Publication 2023-01-21 09:20:06 UTC

R topics documented:

---

board_azure   *Use an Azure storage container as a board*

Description

Pin data to a container in Azure storage using the AzureStor package.

Usage

```
board_azure(
  container,
  path = "",
  n_processes = 10,
  versioned = TRUE,
  cache = NULL
)
```

Arguments

container   An azure storage container created by `AzureStor::blob_container()` or similar.

path   Path to the directory in the container to store pins. Will be created if it doesn't already exist. The equivalent of a `prefix` for AWS S3 storage.

n_processes   Maximum number of processes used for parallel uploads/downloads.

versioned   Should this board be registered with support for versions?

cache   Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.

Details

You can create a board in any of the services that AzureStor supports: blob storage, file storage and Azure Data Lake Storage Gen2 (ADLSgen2).

Blob storage is the classic storage service that is most familiar to people, but is relatively old and inefficient. ADLSgen2 is a modern replacement API for working with blobs that is much faster when working with directories. You should consider using this rather than the classic blob API where possible; see the examples below.

`board_azure()` is powered by the AzureStor package, which is a suggested dependency of pins (not required for pins in general). If you run into errors when deploying content to a server like https://www.shinyapps.io or Connect, add `library(AzureStor)` to your app or document for automatic dependency discovery.

Examples

```
if (requireNamespace("AzureStor")) {
  # Public access board
  url <- "https://pins.blob.core.windows.net/public-data"
  container <- AzureStor::blob_container(url)
  board <- board_azure(container)
  board %>% pin_read("mtcars")
}

## Not run:
# To create a board that you can write to, you'll need to supply one
# of `key`, `token`, or `sas` to AzureStor::blob_container()
# First, we create a board using the classic Azure blob API
```

```
url <- "https://myaccount.blob.core.windows.net/mycontainer"
container <- AzureStor::blob_container(url, sas = "my-sas")
board <- board_azure(container, "path/to/board")
board %>% pin_write(iris)

# ADLSgen2 is a modern, efficient way to access blobs
# - Use 'dfs' instead of 'blob' in the account URL to use the ADLSgen2 API
# - Use the 'storage_container' generic instead of the service-specific
#   'blob_container'
# - We reuse the board created via the blob API above
adls_url <- "https://myaccount.dfs.core.windows.net/mycontainer"
container <- AzureStor::storage_container(adls_url, sas = "my-sas")
board <- board_azure(container, "path/to/board")
board %>% pin_list()
board %>% pin_read("iris")

## End(Not run)
```

---

board_connect   *Use Posit Connect as board*

Description

To use a Posit Connect board, you need to first authenticate. The easiest way to do so is by launching **Tools** - **Global Options** - **Publishing** - **Connect**, and follow the instructions.

You can share pins with others in Posit Connect by changing the viewers of the document to specific users or groups. This is accomplished by opening the new published pin and then changing access under the settings tab. After you've shared the pin, it will be automatically available to others.

Usage

```
board_connect(
  auth = c("auto", "manual", "envvar", "rsconnect"),
  server = NULL,
  account = NULL,
  key = NULL,
  cache = NULL,
  name = "posit-connect",
  versioned = TRUE,
  use_cache_on_failure = is_interactive()
)

board_rsconnect(
  auth = c("auto", "manual", "envvar", "rsconnect"),
  server = NULL,
  account = NULL,
  key = NULL,
  output_files = FALSE,
```

```
    cache = NULL,
    name = "posit-connect",
    versioned = TRUE,
    use_cache_on_failure = is_interactive(),
    versions = deprecated()
)
```

Arguments

auth	There are three ways to authenticate:

> - auth = "manual" uses arguments server and key.
> - auth = "envvar" uses environment variables CONNECT_API_KEY and CONNECT_SERVER.
> - auth = "rsconnect" uses servers registered with the rsconnect package (filtered by server and account, if provided)
>
> The default, auth = "auto", automatically picks between the three options, using "manual" if server and key are provided, "envvar" if both environment variables are set, and "rsconnect" otherwise.

server	For auth = "manual" or auth = 'envvar', the full url to the server, like http://server.posit.co/rsc or https://connect.posit.co/. For auth = 'rsconnect' a host name used to disambiguate Connect accounts, like server.posit.co or connect.posit.co.
account	A user name used to disambiguate multiple Connect accounts.
key	The Posit Connect API key.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
versioned	Should this board be registered with support for versions?
use_cache_on_failure	If the pin fails to download, is it ok to use the last cached version? Defaults to is_interactive() so you'll be robust to poor internet connectivity when exploring interactively, but you'll get clear errors when the code is deployed.
output_files	**[Deprecated]** No longer supported.
versions	Should this board be registered with support for versions?

Public pins

If your Posit Connect instance allows it, you can share a pin publicly by setting the access type to all:

```
board %>% pin_write(my_df, access_type = "all")
```

(You can also do this in Posit Connect by setting "Access" to "Anyone - no login required")

Now anyone can read your pin through board_url():

```
board <- board_url(c(
  numbers = "https://colorado.posit.co/rsc/great-numbers/"
))
board %>% pin_read("numbers")
```

You can find the URL of a pin with pin_browse().

See Also

Other boards: [board_folder](board_folder())(), board_url()

Examples

```
## Not run:
board <- board_connect()
# Share the mtcars with your team
board %>% pin_write(mtcars, "mtcars")

# Download a shared dataset
board %>% pin_read("timothy/mtcars")

## End(Not run)
```

---

board_folder   *Use a local folder as board*

Description

- `board_folder()` creates a board inside a folder. You can use this to share files by using a folder on a shared network drive or inside a DropBox.
- `board_local()` creates a board in a system data directory. It's useful if you want to share pins between R sessions on your computer, and you don't care where the data lives.
- `board_temp()` creates a temporary board that lives in a session specific temporary directory. It will be automatically deleted once the current R session ends. It's useful for examples and tests.

Usage

```
board_folder(path, versioned = FALSE)

board_local(versioned = FALSE)

board_temp(versioned = FALSE)
```

Arguments

path	Path to directory to store pins. Will be created if it doesn't already exist.
versioned	Should this board be registered with support for versions?

See Also

Other boards: board_connect(), board_url()

Examples

```
# session-specific local board
board <- board_temp()
```

---

board_gcs *Use a Google Cloud Storage bucket as a board*

Description

Pin data to a Google Cloud Storage bucket using the googleCloudStorageR package.

Usage

```
board_gcs(bucket, prefix = NULL, versioned = TRUE, cache = NULL)
```

Arguments

bucket	Bucket name. You can only write to an existing bucket, and you can use googleCloudStorageR::gcs_ge here.
prefix	Prefix within this bucket that this board will occupy. You can use this to maintain multiple independent pin boards within a single GCS bucket. Will typically end with / to take advantage of Google Cloud Storage's directory-like handling.
versioned	Should this board be registered with support for versions?
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.

Authentication

board_gcs() is powered by the googleCloudStorageR package which provides several authentication options, as documented in its main vignette. The two main options are to create a service account key (a JSON file) or an authentication token; you can manage either using the gargle package.

Details

- The functions in pins do not create a new bucket. You can create a new bucket from R with googleCloudStorageR::gcs_create_bucket().
- You can pass arguments for googleCloudStorageR::gcs_upload such as predefinedAcl and upload_type through the dots of pin_write().
- board_gcs() is powered by the googleCloudStorageR package, which is a suggested dependency of pins (not required for pins in general). If you run into errors when deploying content to a server like https://www.shinyapps.io or Connect, add requireNamespame(googleCloudStorageR) to your app or document for automatic dependency discovery.

Examples

```
## Not run:
board <- board_gcs("pins-testing")
board %>% pin_write(mtcars)
board %>% pin_read("mtcars")

# A prefix allows you to have multiple independent boards in the same pin.
board_sales <- board_gcs("company-pins", prefix = "sales/")
board_marketing <- board_gcs("company-pins", prefix = "marketing/")
# You can make the hierarchy arbitrarily deep.

# Pass arguments like `predefinedAcl` through the dots of `pin_write`:
board %>% pin_write(mtcars, predefinedAcl = "publicRead")

## End(Not run)
```

---

board_ms365   *Use a OneDrive or Sharepoint document library as a board*

Description

Pin data to a folder in Onedrive or a SharePoint Online document library using the Microsoft365R package.

Usage

```
board_ms365(
  drive,
  path,
  versioned = TRUE,
  cache = NULL,
  delete_by_item = FALSE
)
```

Arguments

drive	A OneDrive or SharePoint document library object, of class `Microsoft365R::ms_drive`.
path	Path to directory to store pins. This can be either a string containing the path-name like "path/to/board", or a `Microsoft365R::ms_drive_item` object pointing to the board path.
versioned	Should this board be registered with support for versions?
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
delete_by_item	Whether to handle folder deletions on an item-by-item basis, rather than deleting the entire folder at once. You may need to set this to TRUE for a board in SharePoint Online or OneDrive for Business, due to document protection policies that prohibit deleting non-empty folders.

Details

Sharing a board in OneDrive (personal or business) is a bit complicated, as OneDrive normally allows only the person who owns the drive to access files and folders. First, the drive owner has to set the board folder as shared with other users, using either the OneDrive web interface or Microsoft365R's ms_drive_item$create_share_link() method. The other users then call board_ms365 with a *drive item object* in the path argument, pointing to the shared folder. See the examples below.

Sharing a board in SharePoint Online is much more straightforward, assuming all users have access to the document library: in this case, everyone can use the same call board_ms365(doclib, "path/to/board"). If you want to share a board with users outside your team, follow the same steps for sharing a board in OneDrive.

board_ms365() is powered by the Microsoft365R package, which is a suggested dependency of pins (not required for pins in general). If you run into errors when deploying content to a server like https://www.shinyapps.io or Connect, add library(Microsoft365R) to your app or document for automatic dependency discovery.

Examples

```
## Not run:
# A board in your personal OneDrive
od <- Microsoft365R::get_personal_onedrive()
board <- board_ms365(od, "myboard")
board %>% pin_write(iris)

# A board in OneDrive for Business
odb <- Microsoft365R::get_business_onedrive(tenant = "mytenant")
board <- board_ms365(odb, "myproject/board")

# A board in a SharePoint Online document library
sp <- Microsoft365R::get_sharepoint_site("my site", tenant = "mytenant")
doclib <- sp$get_drive()
board <- board_ms365(doclib, "general/project1/board")


## Sharing a board in OneDrive:
# First, create the board on the drive owner's side
board <- board_ms365(od, "myboard")

# Next, let other users write to the folder
# - set the expiry to NULL if you want the folder to be permanently available
od$get_item("myboard")$create_share_link("edit", expiry="30 days")

# On the recipient's side: find the shared folder item, then pass it to board_ms365
shared_items <- od$list_shared_items()
board_folder <- shared_items$remoteItem[[which(shared_items$name == "myboard")]]
board <- board_ms365(od, board_folder)

## End(Not run)
```

---

board_s3   *Use an S3 bucket as a board*

Description

Pin data to an S3 bucket, such as on Amazon's S3 service or MinIO, using the paws.storage package.

Usage

```
board_s3(
  bucket,
  prefix = NULL,
  versioned = TRUE,
  access_key = NULL,
  secret_access_key = NULL,
  session_token = NULL,
  credential_expiration = NULL,
  profile = NULL,
  region = NULL,
  endpoint = NULL,
  cache = NULL
)
```

Arguments

bucket	Bucket name. You can only write to an existing bucket.
prefix	Prefix within this bucket that this board will occupy. You can use this to maintain multiple independent pin boards within a single S3 bucket. Will typically end with / to take advantage of S3's directory-like handling.
versioned	Should this board be registered with support for versions?
access_key, secret_access_key, session_token, credential_expiration	
	Manually control authentication. See documentation below for details.
profile	Role to use from AWS shared credentials/config file.
region	AWS region. If not specified, will be read from `AWS_REGION`, or AWS config file.
endpoint	Endpoint to use; usually generated automatically for AWS from `region`. For MinIO, use the full URL (including scheme like `https://`) of your MinIO endpoint.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.

Authentication

board_s3() is powered by the paws package which provides a wide range of authentication options, as documented at https://github.com/paws-r/paws/blob/main/docs/credentials.md. In brief, there are four main options that are tried in order:

- The access_key and secret_access_key arguments to this function. If you have a temporary session token, you'll also need to supply session_token and credential_expiration. (Not recommended since your secret_access_key will be recorded in .Rhistory)

- The AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY env vars. (And AWS_SESSION_TOKEN and AWS_CREDENTIAL_EXPIRATION env vars if you have a temporary session token)

- The AWS shared credential file, ~/.aws/credentials:

  [profile-name]
  aws_access_key_id=your AWS access key
  aws_secret_access_key=your AWS secret key

  The "default" profile will be used if you don't supply the access key and secret access key as described above. Otherwise you can use the profile argument to use a profile of your choice.

- Automatic authentication from EC2 instance or container IAM role.

See the paws documentation for more unusual options including getting credentials from a command line process, picking a role when running inside an EC2 instance, using a role from another profile, and using multifactor authentication.

Details

- The functions in pins do not create a new bucket. You can create a new bucket from R with paws.

- Some functions like pin_list() will work for an S3 board, but don't return useful output.

- You can pass arguments for paws.storage::s3_put_object such as Tagging and ServerSideEncryption through the dots of pin_write().

- board_s3() is powered by the paws.storage package, which is a suggested dependency of pins (not required for pins in general). If you run into errors when deploying content to a server like https://www.shinyapps.io or Connect, add library(paws.storage) to your app or document for automatic dependency discovery.

Examples

```
## Not run:
board <- board_s3("pins-test-hadley", region = "us-east-2")
board %>% pin_write(mtcars)
board %>% pin_read("mtcars")

# A prefix allows you to have multiple independent boards in the same pin.
board_sales <- board_s3("company-pins", prefix = "sales/")
board_marketing <- board_s3("company-pins", prefix = "marketing/")
# You can make the hierarchy arbitrarily deep.

# Pass arguments like `Tagging` through the dots of `pin_write`:
```

```
board %>% pin_write(mtcars, Tagging = "key1=value1&key2=value2")


## End(Not run)
```

---

board_url   *Use a vector of URLs as a board*

Description

board_url() lets you build up a board from individual urls or a [manifest file].

board_url() is read only.

Usage

```
board_url(urls, cache = NULL, use_cache_on_failure = is_interactive())
```

Arguments

urls   Identify available pins being served at a URL or set of URLs (see details):

- Unnamed string: URL to a [manifest file].
- Named character vector: URLs to specific pins (does not support version-ing).
- Named list: URLs to pin version directories (supports versioning).

cache   Cache path. Every board requires a local cache to avoid downloading files mul-tiple times. The default stores in a standard cache location for your operating system, but you can override if needed.

use_cache_on_failure

If the pin fails to download, is it ok to use the last cached version? Defaults to is_interactive() so you'll be robust to poor internet connectivity when exploring interactively, but you'll get clear errors when the code is deployed.

Details

The way board_url() works depends on the type of the urls argument:

- Unnamed character scalar, i.e. **a single URL** to a [manifest file]: If the URL ends in a /, board_url() will look for a _pins.yaml manifest. If the manifest file parses to a named list, versioning is supported. If it parses to a named character vector, the board will not support versioning.
- **Named character vector of URLs**: If the URLs end in a /, board_url() will look for a data.txt that provides metadata for the associated pin. The easiest way to generate this file is to upload a pin version directory created by [board_folder()]. Versioning is not supported.
- **Named list**, where the values are character vectors of URLs and each element of the vector refers to a version of the particular pin: If a URL ends in a /, board_url() will look for a data.txt that provides metadata. Versioning is supported.

Using a vector of URLs can be useful because `pin_download()` and `pin_read()` will be cached; they'll only re-download the data if it's changed from the last time you downloaded it (using the tools of HTTP caching). You'll also be protected from the vagaries of the internet; if a fresh download fails, you'll get the previously cached result with a warning.

Using a manifest file can be useful because you can serve a board of pins and allow collaborators to access the board straight from a URL, without worrying about board-level storage details.

Some examples are provided in `vignette("using-board-url")`.

See Also

Other boards: `board_connect()`, `board_folder()`

Examples

```
github_raw <- function(x) paste0("https://raw.githubusercontent.com/", x)

## with a named vector of URLs to specific pins:
b1 <- board_url(c(
  files = github_raw("rstudio/pins-r/main/tests/testthat/pin-files/"),
  rds = github_raw("rstudio/pins-r/main/tests/testthat/pin-rds/"),
  raw = github_raw("rstudio/pins-r/main/tests/testthat/pin-files/first.txt")
))

b1 %>% pin_read("rds")
b1 %>% pin_browse("rds", local = TRUE)

b1 %>% pin_download("files")
b1 %>% pin_download("raw")

## with a manifest file:
b2 <- board_url(github_raw("rstudio/pins-r/main/tests/testthat/pin-board/"))
b2 %>% pin_list()
b2 %>% pin_versions("y")
```

cache_browse	*Cache management*

Description

Most boards maintain a local cache so that if you're reading a pin that hasn't changed since the last time you read it, it can be rapidly retrieved from a local cache. These functions help you manage that cache.

- `cache_browse()`: open the cache directory for interactive exploration.
- `cache_info()`: report how much disk space each board's cache uses.
- `cache_prune()`: delete pin versions that you haven't used for days (you'll be asked to confirm before the deletion happens).

In general, there's no real harm to deleting the cached pins, as they'll be re-downloaded as needed. The one exception is legacy_local() which mistakenly stored its pinned data in the cache directory; do not touch this directory.

Usage

```
cache_browse()

cache_info()

cache_prune(days = 30)
```

Arguments

days   Number of days to preserve cached data; any pin versions older than days will
                    be removed.

---

legacy_azure   *Azure board (legacy API)*

Description

To use Microsoft Azure Storage as a board, you'll need an Azure Storage account, an Azure Storage container, and an Azure Storage key. You can sign-up and create those at portal.azure.com.

Usage

```
legacy_azure(
  container = Sys.getenv("AZURE_STORAGE_CONTAINER"),
  account = Sys.getenv("AZURE_STORAGE_ACCOUNT"),
  key = Sys.getenv("AZURE_STORAGE_KEY"),
  cache = NULL,
  name = "azure",
  ...
)

board_register_azure(
  name = "azure",
  container = Sys.getenv("AZURE_STORAGE_CONTAINER"),
  account = Sys.getenv("AZURE_STORAGE_ACCOUNT"),
  key = Sys.getenv("AZURE_STORAGE_KEY"),
  cache = NULL,
  path = NULL,
  ...
)
```

Arguments

container	The name of the Azure Storage container.
account	The name of the Azure Storage account.
key	The access key for the Azure Storage container. You can find this under "Access keys" in your storage account settings.
	The key is equivalent to a password, so generally should not be stored in your script. The easiest alternative is to store it in the AZURE_STORAGE_KEY environment variable, which legacy_azure() will use by default.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
...	Additional parameters required to initialize a particular board.
path	Subdirectory within url

Examples

```
## Not run:
# the following example requires an Azure Storage key
board_register_azure(
  container = "pinscontainer",
  account = "pinsstorage",
  key = "abcabcabcabcabcabcabcabcabcab=="
)

## End(Not run)
```

---

legacy_datatxt   *Remote "data.txt" board (legacy API)*

Description

Use board that for a website that uses the [data.txt](#) specification. A data.txt file is a YAML that provides some basic metadata about a directory of files.

Usage

```
legacy_datatxt(
  url,
  headers = NULL,
  cache = NULL,
  needs_index = TRUE,
  browse_url = url,
  index_updated = NULL,
```

```
  index_randomize = FALSE,
  path = NULL,
  versions = FALSE,
  name = NULL,
  ...
)

board_register_datatxt(url, name = NULL, headers = NULL, cache = NULL, ...)
```

Arguments

url	Path to the `data.txt` file or directory containing it.
headers	Optional list of headers to include or a function to generate them.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
needs_index	Does this board have an index file?
browse_url	Not currently used
index_updated	Callback function used to update index
index_randomize	
	When retrieving `data.txt` at a parameter with random query string to defeat caching?
path	Subdirectory within `url`
versions	Should this board be registered with support for versions?
name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
...	Additional parameters required to initialize a particular board.

Examples

```
# register website board using datatxt file
board_register_datatxt(
  url = "https://datatxt.org/data.txt",
  name = "txtexample",
  cache = tempfile()
)

# find pins
pin_find(board = "txtexample")
```

---

legacy_dospace *DigitalOcean board (legacy API)*

Description

To use DigitalOcean Spaces as a board, you first need an DigitalOcean space and a storage key. You can sign-up and create those at digitalocean.com.

Usage

```
legacy_dospace(
  space = Sys.getenv("DO_SPACE"),
  key = Sys.getenv("DO_ACCESS_KEY_ID"),
  secret = Sys.getenv("DO_SECRET_ACCESS_KEY"),
  datacenter = Sys.getenv("DO_DATACENTER"),
  cache = NULL,
  host = "digitaloceanspaces.com",
  name = "dospace",
  ...
)

board_register_dospace(
  name = "dospace",
  space = Sys.getenv("DO_SPACE"),
  key = Sys.getenv("DO_ACCESS_KEY_ID"),
  secret = Sys.getenv("DO_SECRET_ACCESS_KEY"),
  datacenter = Sys.getenv("DO_DATACENTER"),
  cache = NULL,
  host = "digitaloceanspaces.com",
  path = NULL,
  ...
)
```

Arguments

space	The name of the DigitalOcean space.
key, secret	The key and secret for your space. You can create a key and secret in the "Spaces access keys" in your API settings.
	The secret is equivalent to a password, so generally should not be stored in your script. The easiest alternative is to store it in the DO_SECRET_ACCESS_KEY environment variable, which legacy_dospace() will use by default.
datacenter	The datacenter name.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
host	The host to use for storage, defaults to "digitaloceanspaces.com".

name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
...	Additional parameters required to initialize a particular board.
path	Subdirectory within `url`

Examples

```
## Not run:
# the following example requires a DigitalOcean Spaces API key
board <- legacy_dospace(bucket = "s3bucket")

## End(Not run)
```

---

legacy_gcloud   *Google Cloud board (legacy API)*

Description

To use a Google Cloud Storage board, you first need a Google Cloud Storage account, a Google Storage bucket, and an access token or the Google Cloud SDK properly installed and configured. You can sign-up and create these from https://console.cloud.google.com

Usage

```
legacy_gcloud(
  bucket = Sys.getenv("GCLOUD_STORAGE_BUCKET"),
  token = NULL,
  cache = NULL,
  name = "gcloud",
  ...
)

board_register_gcloud(
  name = "gcloud",
  bucket = Sys.getenv("GCLOUD_STORAGE_BUCKET"),
  token = NULL,
  cache = NULL,
  path = NULL,
  ...
)
```

Arguments

bucket	The name of the Google Cloud Storage bucket. Defaults to the `GCLOUD_STORAGE_BUCKET` environment variable.

token	The access token of the Google Cloud Storage container. Generally, it's best to leave this as `NULL`, and rely on the installed Google Cloud SDK to handle authentication.
	If you do want to use an access token, you can retrieve it from https://developers.google.com/oauthplayground. You will need to authorize the "Google Storage API v1" scope.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
...	Additional parameters required to initialize a particular board.
path	Subdirectory within `url`

Examples

```
## Not run:
# the following example requires the Google Cloud SDK to be configured
board <- legacy_gcloud(container = "gcloudcontainer")

## End(Not run)
```

---

legacy_github   *GitHub board (legacy API)*

Description

To use a GitHub board, you'll need to set up authentication, following the instructions at https://happygitwithr.com/https-pat.html#https-pat.

Usage

```
legacy_github(
  repo,
  branch = NULL,
  token = NULL,
  path = "",
  host = "https://api.github.com",
  name = "github",
  cache = NULL,
  ...
)

board_register_github(
  name = "github",
  repo = NULL,
```

```
    branch = NULL,
    token = NULL,
    path = "",
    host = "https://api.github.com",
    cache = NULL,
    ...
)
```

Arguments

repo	The GitHub repository formatted as 'owner/repo'.
branch	The branch to use to commit pins. Default, NULL, will use `main` or `master` if present.
token	GitHub personal access token. Uses [gitcreds](#) if not set.
path	The subdirectory in the repo where the pins will be stored.
host	The URL of the GitHub API. You'll need to customise this to use GitHub enterprise, e.g. `"https://yourhostname/api/v3"`.
name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.
...	Additional parameters required to initialize a particular board.

Large Files

A GitHub repo only supports files under 25MB in size (100MB in theory but there is additional overhead when using the GitHub API). To store large files, GitHub [recommends](#) storing them using GitHub Releases which support up to 2GB files, which is what pins uses. You don't need to do anything extra as this will happen behind the scenes, but don't be surprised if pins creates releases in your repo.

Examples

```
## Not run:
# the following example requires a GitHub API key
board <- legacy_github("owner/repo")

## End(Not run)
```

---

legacy_local   *Local board (legacy API)*

Description

legacy_local() powers board_register_local(), which allows you to access local pins created in earlier versions of the pins package. For new pins, we recommend that you transition to [board_local()](#) which supports the new pins API.

legacy_temp() creates a legacy board in a temporary location, for use in tests and examples.

Usage

```
legacy_local(path = NULL, name = "local", versions = FALSE)

board_register_local(name = "local", cache = NULL, ...)

legacy_temp(name = "temp", ...)
```

Arguments

path
: Path where pins will be stored. If not supplied, defaults to a system **cache** directory, which may be deleted by the operating system if you run out of disk space.

name
: An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.

versions
: Should this board be registered with support for versions?

cache
: Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.

...
: Additional parameters required to initialize a particular board.

Examples

```
# Old api
pin(data.frame(x = 1:3), "test")
pin_get("test")

# New api
board <- board_local()
board %>% pin_write(data.frame(x = 1:3), "test")
board %>% pin_read("test")
```

---

legacy_s3   *S3 board (legacy API)*

Description

To use an Amazon S3 Storage board, you need an Amazon S3 bucket and a user with enough permissions to access the S3 bucket. You can sign-up and create those at https://aws.amazon.com/. Note that it can take a few minutes after you've created it before a bucket is usable.

See board_s3() for a modern version of this legacy board.

Usage

```
legacy_s3(
  bucket = Sys.getenv("AWS_BUCKET"),
  key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  cache = NULL,
  region = NULL,
  host = "s3.amazonaws.com",
  name = "s3",
  ...
)

board_register_s3(
  name = "s3",
  bucket = Sys.getenv("AWS_BUCKET"),
  key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  cache = NULL,
  host = "s3.amazonaws.com",
  region = NULL,
  path = NULL,
  ...
)
```

Arguments

bucket	The name of the Amazon S3 bucket.
key, secret	The key and secret for your space. You can create a key and secret in the "Spaces access keys" in your API settings.
	The secret is equivalent to a password, so generally should not be stored in your script. The easiest alternative is to store it in the AWS_SECRET_ACCESS_KEY environment variable, which board_s3() will use by default.
cache	Cache path. Every board requires a local cache to avoid downloading files multiple times. The default stores in a standard cache location for your operating system, but you can override if needed.

region	The region to use, required in some AWS regions and to enable V4 signatures.
host	The host to use for storage, defaults to `"s3.amazonaws.com"`.
name	An optional name used identify the board. This is no longer generally needed since you should be passing around an explicit board object.
...	Additional parameters required to initialize a particular board.
path	Subdirectory within `url`

Examples

```
## Not run:
# the following example requires an Amazon S3 API key
board <- legacy_s3(bucket = "s3bucket")

## End(Not run)
```

| pin | *Pin a resource (legacy API)* |

Description

Pins the given resource locally or to the given board.

Usage

```
pin(x, name = NULL, description = NULL, board = NULL, ...)
```

Arguments

x	An object, local file or remote URL to pin.
name	The name for the dataset or object.
description	Optional description for this pin.
board	The board where this pin will be placed.
...	Additional parameters.

Details

`pin()` allows you to cache remote resources and intermediate results with ease. When caching remote resources, usually URLs, it will check for HTTP caching headers to avoid re-downloading when the remote result has not changed.

This makes it ideal to support reproducible research by requiring manual instruction to download resources before running your R script.

In addition, `pin()` still works when working offline or when the remote resource becomes unavailable; when this happens, a warning will be triggered but your code will continue to work.

`pin()` stores data frames in two files, an R native file (RDS) and a 'CSV' file. To force saving a pin in R's native format only, you can use `pin(I(data))`. This can improve performance and size at the cost of making the pin unreadable from other tools and programming languages.

Examples

```
# old API
board_register_local(cache = tempfile())
pin(mtcars)
pin_get("mtcars")

# new api
board <- board_local()
board %>% pin_write(mtcars)
board %>% pin_read("mtcars")
```

---

pin_browse   *Browse source of a pin*

Description

pin_browse() navigates you to the home of a pin, either on the internet or on your local file system.

Usage

```
pin_browse(board, name, version = NULL, local = FALSE)
```

Arguments

board	A pin board, created by [board_folder()](), [board_connect()](), [board_url()]() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use [pin_versions()]() to find out which versions are available and when they were created.
local	If TRUE, will open the local copy of the pin; otherwise will show you the home of the pin on the internet.

Examples

```
board <- board_temp(versioned = TRUE)
board %>% pin_write(1:10, "x")
board %>% pin_write(1:11, "x")
board %>% pin_write(1:12, "x")

board %>% pin_browse("x", local = TRUE)
```

---

pin_delete   *Delete a pin*

Description

Delete a pin (or pins), removing it from the board

Usage

```
pin_delete(board, names, ...)
```

Arguments

board	A pin board, created by [board_folder()](), [board_connect()](), [board_url()]() or another board_ function.
names	The names of one or more pins to delete
...	Additional arguments passed on to methods for a specific board.

Examples

```
board <- board_temp()
board %>% pin_write(1:5, "x")
board %>% pin_write(mtcars)
board %>% pin_write(runif(1e6), "y")
board %>% pin_list()

board %>% pin_delete(c("x", "y"))
board %>% pin_list()
```

---

pin_download   *Upload and download files to and from a board*

Description

This is a lower-level interface than pin_read() and pin_write() that you can use to pin any file, as opposed to any R object. The path returned by pin_download() is a read-only path to a cached file: you should never attempt to modify this file.

Usage

```
pin_download(board, name, version = NULL, hash = NULL, ...)

pin_upload(
  board,
  paths,
  name = NULL,
```

```
  title = NULL,
  description = NULL,
  metadata = NULL,
  ...
)
```

Arguments

board	A pin board, created by [board_folder()](), [board_connect()](), [board_url()]() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use [pin_versions()]() to find out which versions are available and when they were created.
hash	Specify a hash to verify that you get exactly the dataset that you expect. You can find the hash of an existing pin by looking for pin_hash in [pin_meta()]().
...	Additional arguments passed on to methods for a specific board.
paths	A character vector of file paths to upload to board.
title	A title for the pin; most important for shared boards so that others can understand what the pin contains. If omitted, a brief description of the contents will be automatically generated.
description	A detailed description of the pin contents.
metadata	A list containing additional metadata to store with the pin. When retrieving the pin, this will be stored in the user key, to avoid potential clashes with the metadata that pins itself uses.

Value

pin_download() returns a character vector of file paths; pin_upload() returns the fully qualified name of the new pin, invisibly.

Examples

```
board <- board_temp()

board %>% pin_upload(system.file("CITATION"))
path <- board %>% pin_download("CITATION")
path
readLines(path)[1:5]
```

---

pin_exists   *Determine if a pin exists*

Description

Determine if a pin exists

Usage

```
pin_exists(board, name, ...)
```

Arguments

board	A pin board, created by [board_folder()](), [board_connect()](), [board_url()]() or another board_ function.
name	Pin name.
...	Additional arguments passed on to methods for a specific board.

---

pin_find   *Search for pins (legacy API)*

Description

Search for pins in legacy boards.

Usage

```
pin_find(
  text = NULL,
  board = NULL,
  name = NULL,
  extended = FALSE,
  metadata = FALSE,
  ...
)
```

Arguments

text	The text to find in the pin description or name.
board	The board name used to find the pin.
name	The exact name of the pin to match when searching.
extended	Should additional board-specific columns be shown?
metadata	Include pin metadata in results?
...	Additional parameters.

Examples

```
pin_find("cars")
# ->
board <- board_local()
board %>% pin_search("cars")
```

---

pin_get   *Retrieve a pin (legacy API)*

Description

Retrieves a pin by name from the local or given board.

Usage

```
pin_get(
  name,
  board = NULL,
  cache = TRUE,
  extract = NULL,
  version = NULL,
  files = FALSE,
  signature = NULL,
  ...
)
```

Arguments

name	The name of the pin.
board	The board where this pin will be retrieved from.
cache	Should the pin cache be used? Defaults to `TRUE`.
extract	Should compressed files be extracted? Each board defines the default behavior.
version	The version of the dataset to retrieve, defaults to latest one.
files	Should only the file names be returned?
signature	Optional signature to validate this pin, use `pin_info()` to compute signature.
...	Additional parameters.

Details

`pin_get()` retrieves a pin by name and, by default, from the local board. You can use the board parameter to specify which board to retrieve a pin from. If a board is not specified, it will use `pin_find()` to find the pin across all boards and retrieve the one that matches by name.

Examples

```
# define temporary board
board <- legacy_temp()
pin(mtcars, board = board)

# retrieve the mtcars pin
pin_get("mtcars", board = board)
```

---

pin_info	*Retrieve pin metadata (legacy API)*

Description

Retrieve metadata for pins in legacy boards.

Usage

```
pin_info(
  name,
  board = NULL,
  extended = TRUE,
  metadata = TRUE,
  signature = FALSE,
  ...
)
```

Arguments

name	The exact name of the pin to match when searching.
board	The board name used to find the pin.
extended	Should additional board-specific information be shown?
metadata	Should additional pin-specific information be shown?
signature	Should a signature to identify this pin be shown?
...	Additional parameters.

Examples

```
# old API
board_register_local(cache = tempfile())
pin(mtcars)
pin_info("mtcars", "local")

# new API
board <- board_temp()
board %>% pin_write(mtcars)
board %>% pin_meta("mtcars")
```

`pin_list`   *List all pins*

Description

List names of all pins in a board. This is a low-level function; use `pin_search()` to get more data
about each pin in a convenient form.

Usage

```
pin_list(board, ...)
```

Arguments

board	A pin board, created by `board_folder()`, `board_connect()`, `board_url()` or another `board_` function.
...	Other arguments passed on to methods

Value

A character vector

Examples

```
board <- board_temp()

board %>% pin_write(1:5, "x")
board %>% pin_write(letters, "y")
board %>% pin_write(runif(20), "z")

board %>% pin_list()
```

`pin_meta`   *Retrieve metadata for a pin*

Description

Pin metadata comes from three sources:

- Standard metadata added by `pin_upload()`/`pin_write()`. This includes:
    - `$name` - the pin's name.
    - `$file` - names of files stored in the pin.
    - `$file_size` - size of each file.
    - `$pin_hash` - hash of pin contents.
    - `$type` - type of pin: "rds", "csv", etc

- – `$title` - pin title
- – `$description` - pin description
- – `$tags` - pin tags
- – `$created` - date this (version of the pin) was created
- – `$api_version` - API version used by pin
- Metadata supplied by the user, stored in `$user`. This is untouched from what is supplied in `pin_write()`/`pin_upload()` except for being converted to and from to YAML.
- Local metadata generated when caching the pin, stored in `$local`. This includes information like the version of the pin, and the path its local cache.

Usage

```
pin_meta(board, name, version = NULL, ...)
```

Arguments

board	A pin board, created by `board_folder()`, `board_connect()`, `board_url()` or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use `pin_versions()` to find out which versions are available and when they were created.
...	Additional arguments passed on to methods for a specific board.

Value

A list.

Examples

```
b <- board_temp()
b %>% pin_write(head(mtcars), "mtcars", metadata = list("Hadley" = TRUE))

# Get the pin
b %>% pin_read("mtcars")
# Get its metadata
b %>% pin_meta("mtcars")
# Get path to underlying data
b %>% pin_download("mtcars")

# Use tags instead
b %>% pin_write(head(mtcars), "mtcars", tags = c("fuel-efficiency", "automotive"))
b %>% pin_meta("mtcars")
```

---

pin_reactive　　　　　　　*Reactive Pin (legacy API)*

Description

Creates a pin that reacts to changes in the given board by polling pin_get(), useful when used from the shiny package.

Usage

```
pin_reactive(name, board, interval = 5000, session = NULL, extract = NULL)
```

Arguments

name	The name of the pin.
board	The board where this pin will be retrieved from.
interval	Approximate number of milliseconds to wait to retrieve updated pin. This can be a numeric value, or a function that returns a numeric value.
session	The user session to associate this file reader with, or NULL if none. If non-null, the reader will automatically stop when the session ends.
extract	Should compressed files be extracted? Each board defines the deefault behavior.

---

pin_reactive_read　　　　　　*Wrap a pin in a reactive expression*

Description

pin_reactive_read() and pin_reactive_download() wrap the results of [pin_read()](#) and [pin_download()](#) into a Shiny reactive. This allows you to use pinned data within your app, and have the results automatically recompute when the pin is modified.

Usage

```
pin_reactive_read(board, name, interval = 5000)

pin_reactive_download(board, name, interval = 5000)
```

Arguments

board	A pin board, created by [board_folder()](#), [board_connect()](#), [board_url()](#) or another board_ function.
name	Pin name.
interval	Approximate number of milliseconds to wait between re-downloading the pin metadata to check if anything has changed.

Examples

```
if (FALSE) {
  library(shiny)
  ui <- fluidPage(
    tableOutput("table")
  )

  server <- function(input, output, session) {
    board <- board_local()
    data <- pin_reactive_read(board, "shiny", interval = 1000)
    output$table <- renderTable(data())
  }
  shinyApp(ui, server)
}
```

---

pin_read   *Read and write objects to and from a board*

Description

Use pin_write() to pin an object to board, and pin_read() to retrieve it.

Usage

```
pin_read(board, name, version = NULL, hash = NULL, ...)

pin_write(
  board,
  x,
  name = NULL,
  type = NULL,
  title = NULL,
  description = NULL,
  metadata = NULL,
  versioned = NULL,
  tags = NULL,
  ...
)
```

Arguments

board	A pin board, created by [board_folder()](#), [board_connect()](#), [board_url()](#) or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use [pin_versions()](#) to find out which versions are available and when they were created.

hash	Specify a hash to verify that you get exactly the dataset that you expect. You can find the hash of an existing pin by looking for pin_hash in [pin_meta()]().
...	Additional arguments passed on to methods for a specific board.
x	An object (typically a data frame) to pin.
type	File type used to save x to disk. Must be one of "csv", "json", "rds", "arrow", or "qs". If not supplied, will use JSON for bare lists and RDS for everything else. Be aware that CSV and JSON are plain text formats, while RDS, Arrow, and qs are binary formats.
title	A title for the pin; most important for shared boards so that others can understand what the pin contains. If omitted, a brief description of the contents will be automatically generated.
description	A detailed description of the pin contents.
metadata	A list containing additional metadata to store with the pin. When retrieving the pin, this will be stored in the user key, to avoid potential clashes with the metadata that pins itself uses.
versioned	Should the pin be versioned? The default, NULL, will use the default for board
tags	A character vector of tags for the pin; most important for discoverability on shared boards.

Details

pin_write() takes care of the details of serialising an R object to disk, controlled by the type argument. See [pin_download()]()/[pin_upload()]() if you want to perform the serialisation yourself and work just with files.

Value

pin_read() returns an R object read from the pin; pin_write() returns the fully qualified name of the new pin, invisibly.

Examples

```
b <- board_temp(versioned = TRUE)

b %>% pin_write(1:10, "x", description = "10 numbers")
b

b %>% pin_meta("x")
b %>% pin_read("x")

# Add a new version
b %>% pin_write(2:11, "x")
b %>% pin_read("x")

# Retrieve an older version
b %>% pin_versions("x")
b %>% pin_read("x", version = .Last.value$version[[1]])
# (Normally you'd specify the version with a string, but since the
# version includes the date-time I can't do that in an example)
```

---

pin_remove   *Delete a pin (legacy API)*

Description

Deletes pins from a legacy board.

Usage

```
pin_remove(name, board = NULL)
```

Arguments

name	The name for the pin.
board	The board from where this pin will be removed.

Examples

```
# old API
board_register_local(cache = tempfile())
pin(mtcars)
pin_remove("mtcars")

# new API
board <- board_local()
board %>% pin_write(mtcars)
board %>% pin_delete("mtcars")
```

---

pin_search   *Search for pins*

Description

The underlying search method depends on the board, but most will search for text in the pin name and title.

Usage

```
pin_search(board, search = NULL, ...)
```

Arguments

board	A pin board, created by [board_folder()](), [board_connect()](), [board_url()]() or another board_ function.
search	A string to search for in pin name and title. Use NULL to return all pins.
...	Additional arguments passed on to methods.

Value

A data frame that summarises the metadata for each pin. Key attributes (name, `type`, `description`, `created`, and `file_size`) are pulled out into columns; everything else can be found in the `meta` list-column.

Examples

```
board <- board_temp()

board %>% pin_write(1:5, "x", title = "Some numbers")
board %>% pin_write(letters[c(1, 5, 10, 15, 21)], "y", title = "My favourite letters")
board %>% pin_write(runif(20), "z", title = "Random numbers")

board %>% pin_search()
board %>% pin_search("number")
board %>% pin_search("letters")
```

---

pin_versions   *List, delete, and prune pin versions*

Description

- `pin_versions()` lists available versions a pin.
- `pin_versions_prune()` deletes old versions.
- `pin_version_delete()` deletes a single version.

Usage

```
pin_versions(board, name, ..., full = deprecated())

pin_version_delete(board, name, version, ...)

pin_versions_prune(board, name, n = NULL, days = NULL, ...)
```

Arguments

board, name	A pair of board and pin name. For modern boards, use board %>% pin_versions(name). For backward compatibility with the legacy API, you can also use pin_versions(name) or pin_version(name, board).
...	Additional arguments passed on to methods for a specific board.
full	**[Deprecated]**
version	Version identifier.
n, days	Pick one of n or days to choose how many versions to keep. n = 3 will keep the last three versions, days = 14 will keep all the versions in the 14 days. Regardless of what values you set, pin_versions_prune() will never delete the most recent version.

Value

A data frame with at least a `version` column. Some boards may provided additional data.

Examples

```
board <- board_temp(versioned = TRUE)

board %>% pin_write(data.frame(x = 1:5), name = "df")
board %>% pin_write(data.frame(x = 2:6), name = "df")
board %>% pin_write(data.frame(x = 3:7), name = "df")

# pin_read() returns the latest version by default
board %>% pin_read("df")

# but you can return earlier versions if needed
board %>% pin_versions("df")

ver <- pin_versions(board, "df")$version[[1]]
board %>% pin_read("df", version = ver)

# delete all versions created more than 30 days ago
board %>% pin_versions_prune("df", days = 30)
```

---

write_board_manifest   *Write board manifest file to board's root directory*

Description

A board manifest file records all the pins, along with their versions, stored on a board. This can be useful for a board built using, for example, [board_folder()](#) or [board_s3()](#), then served as a website, such that others can consume using [board_url()](#). The manifest file is *not* versioned like a pin is, and this function will overwrite any existing `_pins.yaml` file on your board. It is your responsibility as the user to keep the manifest up to date.

Some examples are provided in `vignette("using-board-url")`.

Usage

```
write_board_manifest(board, ...)
```

Arguments

board	A pin board that is *not* read-only.
...	Additional arguments passed on to methods for a specific board.

Details

This function is not supported for read-only boards. It is called for the side-effect of writing a manifest file, _pins.yaml, to the root directory of the board. (This will not work in the unlikely event that you attempt to create a pin called "_pins.yaml".)

The behavior of the legacy API (for example, pin_find()) is unspecified once you have written a board manifest file to a board's root directory. We recommend you only use write_board_manifest() with modern boards.

Value

The board, invisibly

Examples

```
board <- board_temp()
pin_write(board, mtcars, "mtcars-csv", type = "csv")
pin_write(board, mtcars, "mtcars-json", type = "json")

write_board_manifest(board)

# see the manifest's format:
fs::path(board$path, "_pins.yaml") %>% readLines() %>% cat(sep = "\n")

# if you write another pin, the manifest file is out of date:
pin_write(board, 1:10, "nice-numbers", type = "json")

# you decide when to update the manifest:
write_board_manifest(board)
```

Index