Package: orderly1 (via r-universe)

May 21, 2026

Title Lightweight Reproducible Reporting

Version 1.7.2

Description Order, create and store reports from R. By defining a lightweight interface around the inputs and outputs of an analysis, a lot of the repetitive work for reproducible research can be automated. We define a simple format for organising and describing work that facilitates collaborative reproducible research and acknowledges that all analyses are run multiple times over their lifespans.

License MIT + file LICENSE

Encoding UTF-8

URL https://www.vaccineimpact.org/orderly/, https://github.com/vimc/orderly

BugReports https://github.com/vimc/orderly/issues

SystemRequirements git

Imports DBI, R6, RSQLite (>= 2.2.4), crayon, digest, docopt, fs (>= 1.2.7), gert, ids, withr, yaml, zip (>= 2.0.0)

Suggests httr, jsonlite, knitr, markdown, mockery, processx, rmarkdown, testthat, vaultr (>= 1.0.4)

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Language en-GB

Config/pak/sysreqs cmake git make libgit2-dev libuv1-dev libssl-dev

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

Date/Publication 2025-09-19 19:57:23 UTC

RemoteUrl https://github.com/vimc/orderly1

RemoteRef main

RemoteSha da33851ab26ba3acb07874ff8c82008217ad3594

Contents

`orderly_batch` *Run a batch of reports.*

Description

Run one report multiple times with different sets of parameters.

Usage

```
orderly_batch(name = NULL, parameters = NULL, continue_on_error = TRUE, ...)
```

Arguments

name	Name of the report to run (see orderly_list()). A leading src/ will be removed if provided, allowing easier use of autocomplete. Alternatively, the default of NULL is useful if you have already set the working directory to be the source directory.
parameters	Data frame of parameters passed to report. Each row represents a parameter set to be passed to one report run.
continue_on_error	If FALSE then if one report run fails the function will not attempt to run subsequent reports in the batch. If TRUE subsequent parameter sets will be run. If the report run fails during preparation e.g. because of missing parameters this will error and stop all subsequent parameter sets.
...	Additional args passed to orderly_run()

Value

List of ids of newly created reports

See Also

orderly_run() for details of report running

Examples

```
path <- orderly1::orderly_example("demo")
params <- data.frame(nmin = c(0.2, 0.25))
ids <- orderly1::orderly_batch("other", params, root = path)
```

---

orderly_bundle_pack *Pack and run orderly "bundles"*

Description

Pack up and run orderly reports to run elsewhere. By using these functions you can safely copy all requirements of an orderly report into a portable archive and run them on another machine (perhaps a cluster or HPC), then import the completed archive into your orderly tree. There is considerable overhead to using these functions (mostly due to transport costs) so they are intended primarily for very computationally demanding patterns.

Usage

```
orderly_bundle_pack(
  path,
  name,
  parameters = NULL,
  envir = NULL,
  root = NULL,
  locate = TRUE,
  message = NULL,
  instance = NULL,
  remote = NULL,
  tags = NULL
)

orderly_bundle_run(path, workdir = tempfile(), echo = TRUE, envir = NULL)

orderly_bundle_import(path, root = NULL, locate = TRUE)

orderly_bundle_list(path)
```

Arguments

path	A path, whose interpretation depends on the function: orderly_bundle_pack: A directory to save bundles to. If it does not exist it will be created for you. orderly_bundle_run: The path to the packed bundle (a zip file created by orderly_bundle_pack) orderly_bundle_import: The path to unpack and import (a zip file created by orderly_bundle_run) orderly_bundle_list: The path to a directory that might contain either incomplete or complete bundles (created by either orderly_bundle_pack or orderly_bundle_run)
name	Name of the report to pack (see orderly_list(). A leading src/ will be removed if provided, allowing easier use of autocomplete.
parameters	Parameters passed to the report. A named list of parameters declared in the orderly.yml. Each parameter must be a scalar character, numeric, integer or logical.
envir	The parent of the environment that will be used to evaluate the report script; by default a new environment will be made with the global environment as the parent.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
message	An optional character string containing a message explaining why the report was run

instance	Select instance of the source database to be used, where multiple instances are configured. Use a single *unnamed* character string to indicate an instance to match. If given, then this name must be present in all databases where instances are listed in `orderly_config.yml`, and will be ignored by all database where instances are not given. See the "orderly" vignette for further information.
remote	Remote to use to resolve dependencies. Use this in order to run a report with the same dependencies as are available on a remote server, particularly when using `id = "latest"`. Note that this is not the same as running `orderly_pull_dependencies()`, then `orderly_run` with `remote = NULL`, as the pull/run approach will use the latest report in *your* archive but the `remote = "remote"` approach will use the latest approach in the *remote* archive (which might be less recent).
tags	Character vector of tags to add to the report. Tags are immutable and cannot be removed once the report is run. Tags added here will be *in addition* to any tags listed in the `tags:` field in `orderly.yml` and must be present in `orderly_config.yml`.
workdir	The path in which to run bundles. If it does not exist it will be created for you. The completed bundle will be saved in this directory as `<id>.zip`.
echo	Print the result of running the R code to the console

Value

For `orderly_bundle_pack` and `orderly_bundle_run`, a list with elements `path` (the path to the bundle) and `id` (its orderly id). For `orderly_bundle_list` a data.frame with key information about the report in the bundles (id, name, parameters, status, time). The function `orderly_bundle_import` is called for its side effect only and does not return anything useful.

Examples

```
path <- orderly1::orderly_example("minimal")

# A working directory to export bundles to:
workdir <- tempfile()

# Pack up the "example" report to go:
res <- orderly1::orderly_bundle_pack(workdir, "example", root = path)

# The return value is a list with the id and the path to the zip
# file created:
res

# A list of reports bundled in this directory and their status
orderly1::orderly_bundle_list(workdir)

# Run the bundle (this would ordinarily be done on another computer)
zip <- orderly1::orderly_bundle_run(res$path, workdir)
zip

# The status has now been updated to reflect the status
orderly1::orderly_bundle_list(workdir)
```

```
# We can import this into the orderly tree
orderly1::orderly_bundle_import(zip$path, root = path)

# This has now been included in your orderly archive and the
# workdir can be safely deleted
unlink(workdir, recursive = TRUE)
orderly1::orderly_list_archive(path)
```

`orderly_bundle_pack_remote` *Pack and import bundles with remotes*

Description

Pack a bundle on a remote. This is like calling `orderly_bundle_pack()` on the remote and can be used to extract a long-running report from a server to run (say) on an HPC system.

Usage

```
orderly_bundle_pack_remote(
  name,
  parameters = NULL,
  instance = NULL,
  root = NULL,
  locate = TRUE,
  remote = NULL,
  dest = tempdir()
)

orderly_bundle_import_remote(path, root = NULL, locate = TRUE, remote = NULL)
```

Arguments

`name`	Name of the report to pack (see `orderly_list()`. A leading `src/` will be removed if provided, allowing easier use of autocomplete.
`parameters`	Parameters passed to the report. A named list of parameters declared in the `orderly.yml`. Each parameter must be a scalar character, numeric, integer or logical.
`instance`	Select instance of the source database to be used, where multiple instances are configured. Use a single *unnamed* character string to indicate an instance to match. If given, then this name must be present in all databases where instances are listed in `orderly_config.yml`, and will be ignored by all database where instances are not given. See the "orderly" vignette for further information.
`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.

`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.
`remote`	The remote to pack the bundle from, or import into
`dest`	Optional path to write bundle to (a directory name). By default we use the temporary directory and return the full path to the created file.
`path`	The path to unpack and import (a zip file created by `orderly_bundle_run`)

Details

The workflow here will typically be:

1. Use `orderly_bundle_pack_remote()` to create a local copy of a bundle, extracted from a remote. Typically this will be run from the system where the bundle will be run (an HPC head-node or another powerful computer).
2. Run the bundle using `orderly_bundle_run()`
3. Re-import the completed bundle using `orderly_bundle_import_remote` which sends the zip file to the remote and adds it to the archive.

Typically these commands will *not* be run from the orderly root. However, the `root` argument may still be used to find your remote configuration. Alternatively, if your `remote` argument is an orderly remote (e.g., `orderly_remote_path()`, or `orderlyweb`'s `orderlyweb::orderlyweb_remote`) then the `root` and `locate` arguments will be ignored and this command can be run from anywhere. This is the recommended configuration for running on a HPC system.

`orderly_cancel_remote` *Cancel a report*

Description

The action will depend on the status of the report:

- queued - report run will be deleted
- running - report run will be cancelled
- complete/errored - no effect

Usage

```
orderly_cancel_remote(keys, root = NULL, locate = TRUE, remote = NULL)
```

Arguments

keys	The key or keys for the reports to cancel
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
remote	Description of the location. Typically this is a character string indicating a remote specified in the remotes block of your orderly_config.yml. It is also possible to pass in a directly created remote object (e.g., using orderly_remote_path(), or one provided by another package). If left NULL, then the default remote for this orderly repository is used - by default that is the first listed remote.

Value

List with names as report keys and values are lists containing

- killed - boolean TRUE if report successfully cancelled, FALSE otherwise
- message - string detailing reason why cancellation failed

---

orderly_cleanup *Orderly cleanup*

Description

Clean up orderly draft and data directories. Deletes all drafts (possibly just for a set of report names) and then deletes dangling data sets that are not pointed to by any draft or committed reports. Running cleanup does not affect any reports that have been committed with orderly_commit() (i.e., the contents of the archive/ directory).

Usage

```
orderly_cleanup(
  name = NULL,
  root = NULL,
  locate = TRUE,
  draft = TRUE,
  data = TRUE,
  failed_only = FALSE
)
```

Arguments

name	Optional name; in this case only clean up drafts with this name
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
draft	Logical, indicating if drafts should be removed
data	Logical, indicating if dangling data should be removed (data not used by any draft or archived report).
failed_only	Delete only failed reports (those without the end-of-run metadata). This will also clean up drafts created by orderly_test_start()

Value

No return value, this function is called only for its side effects

Examples

```
# In a new example orderly, run two reports and commit only the
# second one:
path <- orderly1::orderly_example("minimal")
id1 <- orderly1::orderly_run("example", root = path)
id2 <- orderly1::orderly_run("example", root = path)
orderly1::orderly_commit(id2, root = path)

# We now have one draft and one archive report:
orderly1::orderly_list_drafts(root = path)
orderly1::orderly_list_archive(root = path)

# To clean up the drafts:
orderly1::orderly_cleanup(root = path)

# We now have no draft and one archive reports:
orderly1::orderly_list_drafts(root = path)
orderly1::orderly_list_archive(root = path)
```

---

orderly_commit *Commit a generated report*

Description

Commit a generated report, moving it from the draft/ directory to archive/ and updating the orderly index. Once committed, reports should not be deleted.

Usage

```
orderly_commit(id, name = NULL, root = NULL, locate = TRUE, timeout = 10)
```

Arguments

id	The identifier of the report
name	The name of the report - this can be omitted and the name will be determined from the id.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
timeout	Time in seconds to wait for db to be available. In parallel the database may become locked so we can choose to wait for timeout seconds before throwing an error.

Value

The path to the newly committed report

Examples

```
# In a new example orderly, run a report
path <- orderly1::orderly_example("minimal")
id <- orderly1::orderly_run("example", root = path)

# To commit it, all we need is the report id
orderly1::orderly_commit(id, root = path)

# The report is now committed, and as such could be used as a
# dependency in another report and is not subject to deletion by
# orderly1::orderly_cleanup
orderly1::orderly_list_archive(root = path)
```

---

orderly_config *Retrieve orderly config object.*

Description

Retrieve orderly config object.

Usage

```
orderly_config(root = NULL, locate = TRUE)
```

Arguments

root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.

Value

An R6 object representing the orderly config.

Public fields

`root` Root dir of the orderly repository

`raw` The raw orderly config yaml

`destination` DB connection configuration for where to store orderly output database. Defaults to local SQLite db `orderly.sqlite`

`fields` Configuration of fields in reports, specifying which are required

`remote` Configuration of remote sources i.e. shared copy of orderly on a remote machine

`vault` Vault server connection information

`global_resources` Path to dir containing global resources.

`changelog` Changelog type configuration

`tags` List of available tags for orderly reports.

`database` Database configuration specifying driver and connection args for (possibly multiple) databases

`archive_version` Orderly version number of the archive

`run_options` List of run options

Methods

Public methods:

- `orderly_config_$new()`
- `orderly_config_$server_options()`
- `orderly_config_$add_run_option()`
- `orderly_config_$get_run_option()`

Method `new()`**:** Create an object representing orderly config

Usage:

```
orderly_config_$new(root, validate = TRUE)
```

Arguments:

`root` Root dir of the orderly repository

`validate` If TRUE migrate cfg to handle any format changes and validate structure if well formed for each of the cfg fields

Method `server_options()`**:** Get connection options for the current server. This is the details from the "remote" section for the server being run on. Server identified via env var `ORDERLY_API_SERVER_IDENTITY`

Usage:

```
orderly_config_$server_options()
```

Returns: Options for current server if can be identified, otherwise NULL

Method `add_run_option()`**:** Add a key-value pair run option

Usage:

```
orderly_config_$add_run_option(name, value)
```

Arguments:

`name` Name of run option

`value` Value for run option

Method `get_run_option()`**:** Retrieve value of a run option

Usage:

```
orderly_config_$get_run_option(name)
```

Arguments:

`name` Name of run option

Examples

```
# The orderly demo, with lots of potential reports:
path <- orderly1::orderly_example("demo")

orderly1::orderly_config(path)
```

`orderly_db` *Connect to orderly databases*

Description

Connect to the orderly databases. These should be treated as as *read-only*.

Usage

```
orderly_db(type, root = NULL, locate = TRUE, validate = TRUE, instance = NULL)
```

Arguments

`type`	The type of connection to make (`source`, `destination`, `csv` or `rds`).
`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.
`validate`	Logical, indicating if the database schema should be validated on open (currently only applicable with `type = "destination"`). This is primarily intended for internal use.
`instance`	Used only by `type = "source"`, and used to select the instance, where multiple instances are configured. Use a single *unnamed* character string to indicate an instance to match. If given, then this name must be present in all databases where instances are listed in `orderly_config.yml`, and will be ignored by all database where instances are not given. See the "orderly" vignette for further information.

Details

Orderly has several databases:

- `source`: All of the databases named in the `database` section of the `orderly_config.yml`
- `destination`: The orderly index database (typically a SQLite database stored at the orderly root)
- `csv`: The cache of database query results, in csv format
- `rds`: The cache of database query results, in rds format

Value

A database connection, or list of connections in the case of `source`.

Examples

```
# Create an orderly that has a single commited report:
path <- orderly1::orderly_example("minimal")
id <- orderly1::orderly_run("example", root = path)
orderly1::orderly_commit(id, root = path)

# The source database holds the data that might be accessible via
# the 'data' entry in orderly.yml:
db <- orderly1::orderly_db("source", root = path)
# This is a list, with one connection per database listed in the
# orderly_config.yml (an empty list if none are specified):
db
DBI::dbListTables(db$source)
head(DBI::dbReadTable(db$source, "data"))
DBI::dbDisconnect(db$source)

# The destination database holds information about the archived
# reports:
db <- orderly1::orderly_db("destination", root = path)
DBI::dbListTables(db)

# These tables are documented online:
# https://vimc.github.io/orderly/schema
DBI::dbReadTable(db, "report_version")
```

---

orderly_deduplicate *Deduplicate an orderly archive*

Description

Deduplicate an orderly archive. Deduplicating an orderly archive will replace all files that have the same content with "hard links". This requires hard link support in the underlying operating system, which is available on all unix-like systems (e.g. MacOS and Linux) and on Windows since Vista. However, on windows systems this might require somewhat elevated privileges. If you use this

feature, it is *very important* that you treat your orderly archive as read-only (though you should be anyway) as changing one copy of a linked file changes all the other instances of it - the files are literally the same file.

Usage

```
orderly_deduplicate(root = NULL, locate = TRUE, dry_run = TRUE, quiet = FALSE)
```

Arguments

root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
dry_run	Logical, indicating if the deduplication should be planned but not run
quiet	Logical, indicating if the status should not be printed

Details

This function will alter your orderly archive. Ordinarily this is not something that should be done, so we try to be careful. In order for this to work, it is *very important* to treat your orderly archive as read-only generally. If your canonical orderly archive is behind OrderlyWeb this will almost certainly be the case already.

With "hard linking", two files with the same content can be updated so that both files point at the same physical bit of data. This is great, as if the file is large, then only one copy needs to be stored. However, this means that if a change is made to one copy of the file, it is immediately reflected in the other, but there is nothing to indicate that the files are linked!

This approach is worth exploring if you have large files that are outputs of one report and inputs to another, or large inputs repeatedly used in different reports, or outputs that end up being the same in multiple reports. If you run the deduplication with dry_run = TRUE, an indication of the savings will be printed.

Value

Invisibly, information about the duplication status of the archive before deduplication was run.

Examples

```
path <- orderly1::orderly_example("demo")
id1 <- orderly1::orderly_run("minimal", root = path)
id2 <- orderly1::orderly_run("minimal", root = path)
orderly_commit(id1, root = path)
orderly_commit(id2, root = path)
tryCatch(
  orderly1::orderly_deduplicate(path, dry_run = TRUE),
  error = function(e) NULL)
```

`orderly_default_remote_set` *Set default remote location*

Description

Set and get default remote locations. Default locations are specific to an orderly repository (based on the path of the repository) so there is no interaction between different orderly projects.

Usage

```
orderly_default_remote_set(value, root = NULL, locate = TRUE)

orderly_default_remote_get(root = NULL, locate = TRUE)
```

Arguments

`value`	A string describing a remote, a remote object, or `NULL` to clear
`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.

Value

The default remote (for `orderly_default_remote_get`). The function `orderly_default_remote_set` is called for its side effects only.

Examples

```
# Same setup as in orderly_remote_path, with a remote orderly:
path_remote <- orderly1::orderly_example("demo")
id <- orderly1::orderly_run("other", list(nmin = 0),
                            root = path_remote, echo = FALSE)
orderly1::orderly_commit(id, root = path_remote)
id <- orderly1::orderly_run("use_dependency",
                            root = path_remote, echo = FALSE)
orderly1::orderly_commit(id, root = path_remote)

# And a local orderly
path_local <- orderly1::orderly_example("demo")

# We'll create an object to interact with this remote using
# orderly_remote_path.
remote <- orderly1::orderly_remote_path(path_remote)

# There is no remote set by default:
```

```
try(orderly1::orderly_default_remote_get(root = path_local))

# We can set one:
orderly1::orderly_default_remote_set(remote, root = path_local)

# and now we can retrieve it:
orderly1::orderly_default_remote_get(root = path_local)

# Note that this has not affected the other orderly:
try(orderly1::orderly_default_remote_get(root = path_remote))
```

`orderly_develop_start` *Develop an orderly report*

Description

The functions `orderly_develop_start`, `orderly_develop_status` and `orderly_develop_clean` provide a workflow for developing a report in much the same way as one might write code outside of orderly. `orderly_develop_start` will copy all files required (global resources and dependencies) into the report source directory, as well as collect all data and parameters - at this point the directory can be developed in directly. It will also load all declared packages, and source all code files listed in the `packages:` and `sources:` sections of your `orderly.yml`. `orderly_develop_status` provides information about the status of files in the directory, while `orderly_develop_clean` deletes all copied files.

Usage

```
orderly_develop_start(
  name = NULL,
  parameters = NULL,
  envir = parent.frame(),
  root = NULL,
  locate = TRUE,
  instance = NULL,
  use_draft = FALSE,
  remote = NULL
)

orderly_develop_status(name = NULL, root = NULL, locate = TRUE)

orderly_develop_clean(name = NULL, root = NULL, locate = TRUE)
```

Arguments

`name`	Name of the report to develop (see `orderly_list()`). A leading `src/` will be removed if provided, allowing easier use of autocomplete. Alternatively, the default of `NULL` is useful if you have already set the working directory to be the source directory.

parameters	Parameters passed to the report. A named list of parameters declared in the orderly.yml. Each parameter must be a scalar character, numeric, integer or logical.
envir	The parent of the environment that will be used to evaluate the report script; by default a new environment will be made with the global environment as the parent.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
instance	Select instance of the source database to be used, where multiple instances are configured. Use a single *unnamed* character string to indicate an instance to match. If given, then this name must be present in all databases where instances are listed in orderly_config.yml, and will be ignored by all database where instances are not given. See the "orderly" vignette for further information.
use_draft	Should draft reports be used for dependencies? This should be used only in development. Valid values are logical (TRUE, FALSE) or use the string newer to use draft reports where they are newer than archive reports. For consistency, always and never are equivalent to TRUE and FALSE, respectively.
remote	Remote to use to resolve dependencies. Use this in order to run a report with the same dependencies as are available on a remote server, particularly when using id = "latest". Note that this is not the same as running orderly_pull_dependencies(), then orderly_run with remote = NULL, as the pull/run approach will use the latest report in *your* archive but the remote = "remote" approach will use the latest approach in the *remote* archive (which might be less recent).

Details

These functions are designed to work within a report's src directory. For example, for a report analysis they will alter or report on the directory src/analysis. It is intended that orderly_develop_start can be run repeatedly; doing this will *refresh* the contents of the directory if upstream files have been updated.

Some degree of care should be used while using these functions.

Because orderly_develop_start copies files into your source tree you should be careful to add these files to your .gitignore files so that they are not included if using git. Rerunning orderly_develop_start will copy a fresh copy of dependencies into your tree, overwriting files that are there without warning.

Repeatedly running orderly_develop_start is "safe", in that it will re-run through the setup steps, but beware that sourcing functions is additive and never subtractive. If you delete (or rename) a function within a source file, it will not be removed from your global environment. Similarly, environment variables will be loaded each time you call this, but no deletions will happen. When in doubt, restart your R session.

Note that these functions are much more permissive as to the state of your orderly.yml than orderly_run() - in particular, they will run, with a message, even if you have not yet defined a script: or any artefacts:.

The orderly_develop_clean function will delete dependencies without warning.

Value

A character vector with the full path to the directory, invisibly.

Examples

```
path <- orderly1::orderly_example("demo")

# This report uses a dependency - it requires that the file
# incoming.csv exists.  This file is created from the report 'other'
orderly1::orderly_develop_status("use_dependency", root = path)

# Copy the required dependencies over, in this case from a draft report
orderly1::orderly_run("other", list(nmin = 0), root = path, echo = FALSE)
orderly1::orderly_develop_start("use_dependency", root = path,
                                use_draft = TRUE)

# Files have been copied across into the source directory
orderly1::orderly_develop_status("use_dependency", root = path)

# The report can then be developed as needed, interactively.  After
# we're happy things can be cleaned up with
orderly1::orderly_develop_clean("use_dependency", root = path)
```

---

orderly_example *Set up an orderly example*

Description

Set up one of the orderly examples included with the package. These are not intended to be starting points for new orderly repositories, but are used in the package examples and vignettes.

Usage

```
orderly_example(
  name,
  path = tempfile(),
  run_demo = FALSE,
  quiet = FALSE,
  git = FALSE
)
```

Arguments

name	Name of the example
path	Destination to create the example - if it exists already it must be an empty directory. By default, creates a new temporary directory
run_demo	Logical, indicating if the example is configured as a "demo" (i.e., with a set of reports to be run and committed), should these be run?
quiet	Logical, indicating if informational messages should be suppressed when running the demo.
git	Logical, indicating if we should create an basic git repository along with the demo. This will have the default orderly .gitignore set up, and a remote which is itself (so that git pull and git fetch run without error, though they will do nothing).

Value

Returns the path to the orderly example

Examples

```
# Create a new copy of the "minimal" example
path <- orderly1::orderly_example("minimal")
dir(path)

# Example reports within this repository:
orderly1::orderly_list(path)
```

---

orderly_graph *Print the dependency tree for a given report using orderly log*

Description

Investigate the dependency structure in a set of orderly reports. This function allows the dependency graph to be created for set of reports that have been run and committed (the archive) or of a set of reports that could be run (the src) to be discovered and printed to screen. *This is experimental and somewhat subject to change and improvement.*

Usage

```
orderly_graph(
  name,
  id = "latest",
  root = NULL,
  locate = TRUE,
  direction = "downstream",
  propagate = TRUE,
  max_depth = Inf,
```

```
  recursion_limit = 100,
  show_all = FALSE,
  use = "archive"
)
```

Arguments

name	the name of the report
id	the id of the report, if omitted, use the id of the latest report
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
direction	A string indicating if we want to move up or down the tree permitted values are upstream, downstream
propagate	A boolean indicating if we want to propagate out of date through the tree
max_depth	A numeric, how far back should the tree go, this can be useful to truncate a very large tree. (default = Inf)
recursion_limit	A numeric, limit for depth of tree, if the tree goes beyond this then an error is thrown. (default = 100)
show_all	A boolean, should we show all reports in the tree, not just the latest.
use	Character string indicating what we read to infer the dependency tree. Current valid values are archive (the default), which reads from archive reports and src which reads from the source reports.

Details

orderly allows a report to rely on the artefacts of one or more other orderly reports. This allows users to develop a network of interconnected reports where the output from report becomes the source of data for another. There are two natural questions that can develop around this workflow:

1. We have updated a report; what are the reports that depend on this so that we can re-run them?
2. We have a report that we want to re-run to ensure uses the latest information. Which other reports are used (directly or indirectly) by this report?

This function displays this information in an easily readable format. Allowing users to see the dependency tree and which reports are out of date and need to be re-run.

Value

An orderly tree object with the root corresponding to the given report.

Remark

By default the tree is built using data from the local report database (see orderly_commit, orderly_db). This means that it will not find changes from a report that has not be run and committed. That is, if a user changes a report to use or create different artefacts this will not be picked up by the function until the reports are re-run and committed to the archive.

It is possible to generate a tree from the source reports by using `use = "src"` - this generates the "theoretical tree", and has no concept of being "up to date" or of ids.

Warning

This interface is considered experimental and may change without notice. Please do not depend on it in scripts as it may break things. Consider this a (hopefully) useful way of exploring the dependencies in your reports *interactively* - let us know what is missing and we'll try and build it out.

Examples

```
path <- orderly1::orderly_example("demo")

id <- orderly1::orderly_run("other", root = path, parameters=list(nmin=0))
orderly1::orderly_commit(id, root = path)
id <- orderly1::orderly_run("use_dependency", root = path)
orderly1::orderly_commit(id, root = path)
id <- orderly1::orderly_run("use_dependency_2", root = path)
orderly1::orderly_commit(id, root = path)
orderly1::orderly_graph("other", root = path)
orderly1::orderly_graph("use_dependency_2", root = path,
                                direction = "upstream")
```

---

orderly_graph_out_of_date

Given a tree return a list of reports to be re-run (and the order that they should be re-run)

Description

Given a tree return a list of reports to be re-run (and the order that they should be re-run)

Usage

```
orderly_graph_out_of_date(tree)
```

Arguments

`tree`	A dependency tree object from orderly_graph_out_of_date

Value

a list of report names to be re-run. First report to rerun first

orderly_info *Return info about a report which has been run*

Description

This will return info from either successful or failed reports. It will look for the report with `id` in archive first and then look in drafts if it can't be found from archive.

Usage

```
orderly_info(id, name, root = NULL, locate = TRUE)
```

Arguments

`id`	The report ID
`name`	The name of the report
`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.

Value

Info from report run - this is subject to change. Returns a list which includes report id, name, indication of success, run date and elapsed time, parameters, git info (if available), path to logfile (if exists) and details of error if the run failed

Examples

```
path <- orderly1::orderly_example("demo")
id <- orderly1::orderly_run("minimal", root = path)
orderly1::orderly_info(id, "minimal", root = path)
```

orderly_init *Initialise an orderly store*

Description

Initialise an orderly store. This is a helper function that automates getting started with using orderly for a new project. It is not required to use - you can create the orderly structure yourself (all that is compulsory is the `orderly_config.yml` file).

Usage

```
orderly_init(root, doc = TRUE, quiet = FALSE)
```

Arguments

root	The root of the store; this must be an empty directory or the path of a directory to create
doc	Logical, indicating if documentation should be added to the directories. This also has the (potentially useful) effect of making these directories noticeable by git.
quiet	Logical, indicating if informational messages should be suppressed.

Details

This function creates a minimal orderly structure, containing:

- `orderly_config.yml`: The orderly configuration. Minimally, this can be empty, but it must exist.
- `src`: The path where report sources live. This should be placed under version control, and contain a number of reports, each in their own directory with an `orderly.yml` describing their inputs and outputs (artefacts). The `orderly_new()` function can be used to accelerate creation of new reports.
- `draft`: A directory where reports will be run using `orderly_run()`. This directory should be excluded from version control. `orderly` will create it as needed if it does not exist when a report is run.
- `archive`: A directory where successfully run reports will be moved to after being committed with `orderly_commit()`. This directory should be excluded from version control. `orderly` will create it as needed if it does not exist when a report is committed.
- `data`: A directory where data extracted from the database (if used) will be stored. This directory should be excluded from version control. `orderly` will create it as needed if it does not exist when a report is run.

Value

The path to the newly created archive

See Also

`orderly_new()` for creating new reports within a configured orderly repository.

Examples

```
# Initialise a new orderly repository in an temporary directory:
path <- orderly1::orderly_init(tempfile())

# This has created the directory skeleton that you need to get
# started using orderly:
fs::dir_tree(path)

# As instructed, the next thing to do is to edit the
# orderly_config.yml file to match your needs:
readLines(file.path(path, "orderly_config.yml"))
```

`orderly_latest` *Find most recent report*

Description

Find most recent version of an orderly report. The most recent report is always the most recently run report that has been committed (regardless of the order in which they were committed).

Usage

```
orderly_latest(
  name = NULL,
  root = NULL,
  locate = TRUE,
  draft = FALSE,
  must_work = TRUE
)
```

Arguments

`name`	Name of the report to find; if `NULL` returns the most recent report across all names
`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.
`draft`	Should draft reports be used searched? Valid values are logical (`TRUE`, `FALSE`) or use the string `newer` to use draft reports where they are newer than archive reports. For consistency, `always` and `never` are equivalent to `TRUE` and `FALSE`, respectively.
`must_work`	Throw an error if no report is found. If `FALSE`, returns `NA_character_`.

Value

A character string with the id of the most recent report

See Also

orderly_list and orderly_list_archive for listing report names and versions.

Examples

```
path <- orderly1::orderly_example("minimal")
id1 <- orderly1::orderly_run("example", root = path, echo = FALSE)
id2 <- orderly1::orderly_run("example", root = path, echo = FALSE)
```

```
# With no reports committed there is no latest report:
orderly1::orderly_latest("example", root = path, must_work = FALSE)

# Commit the first report and it will be reported as latest:
orderly1::orderly_commit(id1, root = path)
orderly1::orderly_latest("example", root = path)

# Commit the second report and it will be reported as latest instead:
orderly1::orderly_commit(id2, root = path)
orderly1::orderly_latest("example", root = path)
```

`orderly_list` *List orderly reports*

Description

List the *names* of reports known to orderly. These are the *source* names, not the results of running reports. Note that if a report has been committed from a different branch it will not appear here, as this is simply the set of reports in the `src` directory that can be run.

Usage

```
orderly_list(root = NULL, locate = TRUE)
```

Arguments

`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.

Value

A character vector of report names

See Also

`orderly_list_archive()` and `orderly_list_drafts()`, which list archived (committed) and draft reports and their versions.

Examples

```
# The orderly demo, with lots of potential reports:
path <- orderly1::orderly_example("demo")

# Reports that _could_ be run:
orderly1::orderly_list(path)
```

`orderly_list_drafts` *List draft and archived reports*

Description

List draft and archived reports. This returns a data.frame with columns `name` (see `orderly_list()`) and `id`.

Usage

```
orderly_list_drafts(root = NULL, locate = TRUE, include_failed = FALSE)

orderly_list_archive(root = NULL, locate = TRUE)
```

Arguments

`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.
`include_failed`	Logical, indicating if failed drafts should be listed (only has an effect for `orderly_list_drafts` as no failed run should make it into the archive). A failed report is one that lacks an `orderly_run.rds` file.

Value

A `data.frame` with columns `name` and `id`, containing character vectors of report names and versions, respectively.

See Also

`orderly_list()`, which lists the names of source reports that can be run, and `orderly_latest()` which returns the id of the most recent report.

Examples

```
# The orderly demo, with lots of potential reports:
path <- orderly1::orderly_example("demo")

# Reports that _could_ be run:
orderly1::orderly_list(path)

# Run a report twice:
id1 <- orderly1::orderly_run("minimal", root = path)
id2 <- orderly1::orderly_run("minimal", root = path)

# We can see both drafts:
```

```
orderly1::orderly_list_drafts(path)

# Nothing is in the archive:
orderly1::orderly_list_archive(path)

# Commit a report:
orderly1::orderly_commit(id2, root = path)

# Only one draft now
orderly1::orderly_list_drafts(path)

# And the second report is in the archive:
orderly1::orderly_list_archive(path)
```

`orderly_list_metadata` *List reports with only local metadata*

Description

List reports that are present only as metadata; these are the result of doing `orderly_pull_archive()` with `recursive = FALSE`, in which case only metadata was downloaded and not the report contents itself.

Usage

```
orderly_list_metadata(root = NULL, locate = FALSE, include_archive = FALSE)
```

Arguments

`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.
`include_archive`	Logical, indicating if we should include reports that are also included in the archive.

Value

A `data.frame()` with columns `name` and `id`, as for `orderly_list_archive()`

Examples

```
path <- orderly1::orderly_example("minimal")
# No metadata-only reports will be present, unless you have run
# orderly1::orderly_pull_archive(..., recursive = FALSE)
orderly1::orderly_list_metadata(path)
```

`orderly_log_on` *Orderly logging and diagnostic messages*

Description

Start and stop the orderly log. When active, some actions will print diagnostic information to the message stream. This is set to be on by default.

Usage

```
orderly_log_on()

orderly_log_off()

orderly_log(topic, value)
```

Arguments

`topic`	Up to 9 character text string with the log topic
`value`	Character string with the log entry

Details

The function `orderly_log` is designed to be used from applications that extend orderly, while the functions `orderly_log_on` and `orderly_log_off` can be used by applications or users to enable and disable log messages.

The interface here may expand by adding arguments or change behaviour based on global options. Future versions may support logging to a file, or adding timestamps, or logging in json format, etc.

Value

`orderly_log_on` and `orderly_log_off` invisibly returns a logical indicating if logging was previously enabled. This allows patterns like:

```
if (!orderly1::orderly_log_off()) {
  on.exit(orderly1::orderly_log_on())
}
```

to disable logging within a function (the `on.exit` block will be run when the function exits).

See Also

`orderly_run()`, which makes use of these log messages

Examples

```
# We are going to log things below
logging_was_enabled <- orderly1::orderly_log_on()

# About orderly log messages:
# Orderly log messages have the form "[title] message"
orderly1::orderly_log_on()
orderly1::orderly_log("title", "message")

# If logging is disabled they are not printed:
orderly1::orderly_log_off()
orderly1::orderly_log("title", "message")

# Restore to previous settings:
if (logging_was_enabled) {
  orderly1::orderly_log_on()
}
```

`orderly_migrate` *Migrate an orderly archive*

Description

Migrate an orderly archive. This is needed periodically when the orderly archive version changes. If you get a message like `orderly archive needs migrating from a.b.c => x.y.z` then you need to run this function. The archive version is at most equal to the package version.

Usage

```
orderly_migrate(
  root = NULL,
  locate = TRUE,
  to = NULL,
  dry_run = FALSE,
  skip_failed = FALSE,
  clean = FALSE
)
```

Arguments

`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.
`to`	The version to migrate to. The default is the current archive version; this is almost always what is wanted.

dry_run	Logical, indicating if we should try running the migration but not actually applying it. This is intended primarily for developing new migrations and will probably not work if you are multiple archive versions behind.
skip_failed	Logical, where TRUE we will skip over entries that failed to be migrated. This is expected to be useful on local archives only because it violates the append-only nature of orderly. However, if a local archive contains unusual copies of orderly archives that can't be migrated this might come in helpful.
clean	Logical, where TRUE (and where the migration was successful and dry_run is FALSE) orderly will clean up all migration backup files. Use this periodically to clean up the archive.

Details

Sometimes we add change information saved out in the orderly run. This requires patching previously run versions of the orderly metadata and that's not something we want to do lightly. This function uses a relatively safe, and reversible, way of migrating metadata. We modify the orderly_run.rds files, but will create versioned backups as files are changed.

Value

No return value, this function is called only for its side effects

Examples

```
# Without an orderly repository created by a previous version of
# orderly, this function does nothing interesting:
path <- orderly1::orderly_example("minimal")
orderly1::orderly_migrate(path)
```

---

orderly_new *Create new report*

Description

Create new report, starting from a template. Orderly comes with a set of templates, but projects can bring their own templates; see Details below for how these are configured and discovered by orderly.

Usage

```
orderly_new(name, root = NULL, locate = TRUE, quiet = FALSE, template = NULL)
```

Arguments

name	Name of the new report (will be a directory name).
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
quiet	Logical, indicating if informational messages should be suppressed.
template	The name of a template. If NULL orderly will search for a template (see Details). If given it must be the name of a directory within a directory template in your project root. The special label "orderly" will use orderly's builtin template.

Details

To create a custom template, create a directory template within your orderly root. Within that directory create directories containing all the files that you would like a report to contain. This *must* contain a file orderly.yml but may contain further files (for example, you might want a default script and Rmd file).

If template is not given (i.e., is NULL) then we look for a template called default (i.e., stored at template/default), then fall back on the system orderly template.

We first look for a file orderly/template.yml within the orderly root. If that is not found, then a copy from the orderly package is used. This can always be used by using template = "system".

Value

The path of the new source directory, invisibly

See Also

orderly_init() for initialising a new orderly repository.

Examples

```
path <- orderly1::orderly_example("minimal")

# Create a new report with the name "myreport" in this orderly
# repository:
orderly1::orderly_new("myreport", root = path)

# The directory will be initialised with a orderly.yml file
# containing documentation
dir(file.path(path, "src", "myreport"))
readLines(file.path(path, "src", "myreport", "orderly.yml"))
```

orderly_packages *Return details of packages required by all src reports in this orderly repo*

Description

Return details of packages required by all src reports in this orderly repo

Usage

```
orderly_packages(root = NULL, locate = TRUE)
```

Arguments

root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.

Value

List of packages required by all src reports

Examples

```
path <- orderly1::orderly_example("minimal")
orderly1::orderly_packages(root = path)
```

orderly_pull_dependencies *Download dependent reports*

Description

Download dependent reports from an orderly remote. This can only be used if the orderly_config.yml lists a remote. This allows for a centralised workflow where a central orderly store exists and holds the canonical copies of reports, from which versions can be downloaded into local stores.

Usage

```
orderly_pull_dependencies(
  name = NULL,
  root = NULL,
  locate = TRUE,
  remote = NULL,
  parameters = NULL,
  recursive = TRUE
)

orderly_pull_archive(
  name,
  id = "latest",
  root = NULL,
  locate = TRUE,
  remote = NULL,
  parameters = NULL,
  recursive = TRUE
)

orderly_push_archive(
  name,
  id = "latest",
  root = NULL,
  locate = TRUE,
  remote = NULL
)
```

Arguments

name	Name of the report to download dependencies for. Alternatively, the default of NULL is useful if you have already set the working directory to be the source directory.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
remote	Description of the location. Typically this is a character string indicating a remote specified in the remotes block of your orderly_config.yml. It is also possible to pass in a directly created remote object (e.g., using orderly_remote_path(), or one provided by another package). If left NULL, then the default remote for this orderly repository is used - by default that is the first listed remote.
parameters	Parameters to pass through when doing dependency resolution. If you are using a query for id that involves a parameter (e.g., latest(parameter:x == p)) you will need to pass in the parameters here. Similarly, if you are pulling a report

that uses query dependencies that reference parameters you need to pass them here (the same parameter set will be passed through to all dependencies).

`recursive` Logical, indicating if all dependencies of a report should also be pulled. Setting this to `FALSE` only the direct reports, along with metadata for the dependencies; this will be potentially much faster, but leaves your archive in a more fragile state.

`id` The identifier (for `orderly_pull_archive`). The default is to use the latest report.

Details

The `orderly_pull_archive` function pulls report directly (without it being a dependent report).

After setting your username up you can run `orderly_pull_dependencies("reportname")` to pull the *dependencies* of `"reportname"` down so that `"reportname"` can be run, or you can run `orderly_pull_archive("reportname")` to pull a copy of `"reportname"` that has been run on the remote server.

Pulling an archive report from a remote also pulls its dependencies (recursively), and adds all of these to the local database. This may require migrating old orderly archives (`orderly_migrate()`). Note that this migration will likely fail for remote orderly versions older than 0.6.8 because the migration needs to read data files on disk that are not included in the downloaded archive in order to collect all the information required for the database. In this case, ask the administrator of the remote orderly archive to migrate their archive, and then re-pull.

Pushing an archive is possible only if the remote supports it. Currently this is supported by `orderly_remote_path()` remotes, though not by orderlyweb remotes. There is no control over what will *accept* a push at this point, nor any check that what you've pushed is "good" except that it exists in your archive. As with pulling an archive, pushes are recursive with respect to dependencies. The configuration interface here will likely change a little over time.

Value

No return value, these functions are called only for their side effects

See Also

`orderly_remote_path()`, which implements the remote interface for orderly repositories at a local path. See also OrderlyWeb for a system for hosting orderly repositories over an HTTP API. `vignette("remote", package = "orderly1")` describes the remote system in more detail.

Examples

```
# Suppose we have a "remote" orderly repository at some path.
# This might be read-only for you in practice and available via a
# network filesystem or a dropbox folder synced to your computer.
# We'll populate this with a pair of reports:
path_remote <- orderly1::orderly_example("demo")
id <- orderly1::orderly_run("other", list(nmin = 0),
                            root = path_remote, echo = FALSE)
orderly1::orderly_commit(id, root = path_remote)
id <- orderly1::orderly_run("use_dependency",
```

```
                            root = path_remote, echo = FALSE)
orderly1::orderly_commit(id, root = path_remote)

# We'll create a an object to interact with this remote using
# orderly_remote_path.
remote <- orderly1::orderly_remote_path(path_remote)

# We can use this object directly
remote$list_reports()
remote$list_versions("other")

# More typically one will interact with the functions
# orderly_pull_archive and orderly_pull_dependencies.

# Now, suppose that you have your "local" copy of this; it shares
# the same source (ordinarily these would both be under version
# control with git):
path_local <- orderly1::orderly_example("demo")

# If we wanted to run the report "use_dependency" we need to have
# a copy of the report "other", on which it depends:
try(orderly1::orderly_run("use_dependency", root = path_local))

# We can "pull" dependencies of a report before running
orderly1::orderly_pull_dependencies("use_dependency", remote = remote,
                                    root = path_local)

# Now we can run the report because we have a local copy of the
# dependency:
orderly1::orderly_run("use_dependency", root = path_local)

# We can also directly pull previously run reports:
orderly1::orderly_pull_archive("use_dependency", id, remote = remote,
                               root = path_local)
orderly1::orderly_list_archive(root = path_local)
```

`orderly_rebuild` *Rebuild the report database*

Description

Rebuild the report database. This is necessary when the orderly database schema changes, and you will be prompted to run this function after upgrading orderly in that case.

Usage

```
orderly_rebuild(
  root = NULL,
  locate = TRUE,
  verbose = TRUE,
```

```
  if_schema_changed = FALSE
)
```

Arguments

root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
verbose	Logical, indicating if information about the rebuild should be printed as it runs
if_schema_changed	Logical, indicating if the rebuild should take place only if the schema has changed. This is designed to be safe to use in (say) deployment scripts because it will be fast enough to call regularly.

Details

The report database (orderly's "destination" database) is essentially an index over all the metadata associated with reports. It is used by orderly itself, and can be used by applications that extend orderly (e.g., OrderlyWeb. All the data in this database can be rebuilt from files stored with the committed (archive) orderly reports, using the orderly_rebuild function.

Value

No return value, this function is called only for its side effects

Examples

```
path <- orderly1::orderly_example("minimal")
id <- orderly1::orderly_run("example", root = path)
orderly1::orderly_commit(id, root = path)

con <- orderly1::orderly_db("destination", root = path)
DBI::dbReadTable(con, "report_version")
DBI::dbDisconnect(con)

# The database can be removed and will be rebuilt if requested
# (this is only a good idea if you do not extend the database with
# your own fields - only the fields that orderly looks after can
# be recovered!)
file.remove(file.path(path, "orderly.sqlite"))
orderly1::orderly_rebuild(path)
file.exists(file.path(path, "orderly.sqlite"))
con <- orderly1::orderly_db("destination", root = path)
DBI::dbReadTable(con, "report_version")
DBI::dbDisconnect(con)

# It is safe to rebuild a database repeatedly, though this can be
# slow with larger databases.
orderly1::orderly_rebuild(path)
```

`orderly_remote` *Get a remote*

Description

Get a remote, based on the configuration in `orderly_config.yml` - different remote drivers have different methods, and this function gives you access to these lower-level objects.

Usage

```
orderly_remote(remote = NULL, root = NULL, locate = TRUE)
```

Arguments

`remote`	Description of the location. Typically this is a character string indicating a remote specified in the `remotes` block of your `orderly_config.yml`. It is also possible to pass in a directly created remote object (e.g., using `orderly_remote_path()`, or one provided by another package). If left `NULL`, then the default remote for this orderly repository is used - by default that is the first listed remote.
`root`	The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.
`locate`	Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.

Value

The orderly remote, as described in `orderly_config.yml` - if no remotes are configured, or if the requested remote does not exist, an error will be thrown.

See Also

`orderly_pull_dependencies()` which provides a higher-level interface to pulling from a remote (including adding the downloaded archive into your orderly repository), and see the documentation underlying the orderly remote driver that your `orderly_config.yml` declares for information about using that remote.

Examples

```
## We need two orderly repositories here - one as a "local" and one as
## a "remote" (see ?orderly_pull_archive)
path_remote <- orderly1::orderly_example("demo")
path_local <- orderly1::orderly_example("demo")

## Configure our remote:
path_config <- file.path(path_local, "orderly_config.yml")
txt <- readLines(path_config)
writeLines(c(
```

```
  txt,
  "remote:",
  "  default:",
  "    driver: orderly1::orderly_remote_path",
  "    args:",
  paste("      path:", path_remote)),
  path_config)

## Get our remote:
remote <- orderly1::orderly_remote(root = path_local)

## Can use the remote's methods to interact directly - actual methods
## depend on the remote driver being used.
remote$list_reports()
```

`orderly_remote_path` *Orderly remote at a different path*

Description

Create a "handle" for interacting with orderly repositories that are hosted at a different path. This might be useful in cases where you have access to an orderly repository via a network mount or a synchronised folder (e.g., Dropbox, Box, etc). More generally, `orderly_remote_path` implements an interface used by orderly to abstract over different ways that orderly repositories might be hosted remotely, including over HTTP APIs.

Usage

```
orderly_remote_path(path, name = NULL)
```

Arguments

`path`	Path to the orderly store
`name`	Name of the remote

Value

An `orderly_remote_path` object, with methods that orderly will use in order to control this remote

See Also

`orderly_pull_dependencies()` and `orderly_pull_archive()`, which are the primary ways these remote objects are used. See also OrderlyWeb for a system for hosting orderly repositories over an HTTP API.

Examples

```
# Suppose we have a "remote" orderly repository at some path.
# This might be read-only for you in practice and available via a
# network filesystem or a dropbox folder synced to your computer.
# We'll populate this with a pair of reports:
path_remote <- orderly1::orderly_example("demo")
id <- orderly1::orderly_run("other", list(nmin = 0),
                            root = path_remote, echo = FALSE)
orderly1::orderly_commit(id, root = path_remote)
id <- orderly1::orderly_run("use_dependency",
                            root = path_remote, echo = FALSE)
orderly1::orderly_commit(id, root = path_remote)

# We'll create a an object to interact with this remote using
# orderly_remote_path.
remote <- orderly1::orderly_remote_path(path_remote)

# We can use this object directly
remote$list_reports()
remote$list_versions("other")

# More typically one will interact with the functions
# orderly_pull_archive and orderly_pull_dependencies.

# Now, suppose that you have your "local" copy of this; it shares
# the same source (ordinarily these would both be under version
# control with git):
path_local <- orderly1::orderly_example("demo")

# If we wanted to run the report "use_dependency" we need to have
# a copy of the report "other", on which it depends:
try(orderly1::orderly_run("use_dependency", root = path_local))

# We can "pull" dependencies of a report before running
orderly1::orderly_pull_dependencies("use_dependency", remote = remote,
                                    root = path_local)

# Now we can run the report because we have a local copy of the
# dependency:
orderly1::orderly_run("use_dependency", root = path_local)

# We can also directly pull previously run reports:
orderly1::orderly_pull_archive("use_dependency", id, remote = remote,
                               root = path_local)
orderly1::orderly_list_archive(root = path_local)
```

`orderly_remote_status` *Get status of remote queue.*

Description

Get the status of the remote queue as a list.

Usage

```
orderly_remote_status(root = NULL, locate = TRUE, remote = NULL)
```

Arguments

root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
remote	Description of the location. Typically this is a character string indicating a remote specified in the remotes block of your orderly_config.yml. It is also possible to pass in a directly created remote object (e.g., using orderly_remote_path(), or one provided by another package). If left NULL, then the default remote for this orderly repository is used - by default that is the first listed remote.

Value

List containing details of running and queued reports on the remote queue. Including report name, status and version (where known)

---

orderly_run *Run a report*

Description

Run a report. This will create a new directory in drafts/<reportname>, copy your declared resources there, extract data from databases (if you are using them), run your script and check that all expected artefacts were created. Once successfully run you can use orderly_commit() to move it to the archive directory.

Usage

```
orderly_run(
  name = NULL,
  parameters = NULL,
  envir = NULL,
  root = NULL,
  locate = TRUE,
  echo = TRUE,
  message = NULL,
  instance = NULL,
```

```
  use_draft = FALSE,
  remote = NULL,
  tags = NULL
)
```

Arguments

name	Name of the report to run (see orderly_list()). A leading src/ will be removed if provided, allowing easier use of autocomplete. Alternatively, the default of NULL is useful if you have already set the working directory to be the source directory.
parameters	Parameters passed to the report. A named list of parameters declared in the orderly.yml. Each parameter must be a scalar character, numeric, integer or logical.
envir	The parent of the environment that will be used to evaluate the report script; by default a new environment will be made with the global environment as the parent.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
echo	Print the result of running the R code to the console
message	An optional character string containing a message explaining why the report was run
instance	Select instance of the source database to be used, where multiple instances are configured. Use a single *unnamed* character string to indicate an instance to match. If given, then this name must be present in all databases where instances are listed in orderly_config.yml, and will be ignored by all database where instances are not given. See the "orderly" vignette for further information.
use_draft	Should draft reports be used for dependencies? This should be used only in development. Valid values are logical (TRUE, FALSE) or use the string newer to use draft reports where they are newer than archive reports. For consistency, always and never are equivalent to TRUE and FALSE, respectively.
remote	Remote to use to resolve dependencies. Use this in order to run a report with the same dependencies as are available on a remote server, particularly when using id = "latest". Note that this is not the same as running orderly_pull_dependencies(), then orderly_run with remote = NULL, as the pull/run approach will use the latest report in *your* archive but the remote = "remote" approach will use the latest approach in the *remote* archive (which might be less recent).
tags	Character vector of tags to add to the report. Tags are immutable and cannot be removed once the report is run. Tags added here will be *in addition* to any tags listed in the tags: field in orderly.yml and must be present in orderly_config.yml.

Details

Parameters are passed to the report as a named list, for example

```
id <- orderly1::orderly_run("other", list(nmin = 0.2), root = path)
```

(see the examples). The names of the parameters (here, `nmin`) must correspond to declared parameters in the `orderly.yml`. It is an error if parameters without a default are omitted, and it is an error if unknown parameters are provided.

Environment variables that are created in `orderly_envir.yml` will be available while the report runs. Those that begin with `ORDERLY_` will be saved into the `orderly_run.rds` within the `$env` section (except for any that match the patterns "TOKEN", "PAT" or "PASS").

Value

The id of the newly created report

See Also

`orderly_log()` for controlling display of log messages (not just R output)

Examples

```
path <- orderly1::orderly_example("demo")

# To run most reports, provide the report name (and the path if
# not running in the working directory, as is the case here):
id <- orderly1::orderly_run("minimal", root = path)

# Every report gets a unique identifier, based on the time (it is
# ISO 8601 time with random hex appended to end)
id

# After being run, a report is a "draft" and will exist in the
# drafts directory:
orderly1::orderly_list_drafts(root = path)

# Draft reports are always stored in the path
# <root>/draft/<name>/<id>, so we have
dir(file.path(path, "draft", "minimal", id))

# which contains the files when the report was run.

# If a report has parameters, then these must be passed in as a
# named list.
id <- orderly1::orderly_run("other", list(nmin = 0.2), root = path)

# These parameters can be used in SQL queries or in the report
# code.
```

`orderly_run_info` *Information on current orderly run*

Description

This function allows inspection of some of orderly's metadata during an orderly run. The format returned is internal to orderly and subject to change. It is designed to be used within report code. To use in conjunction with `orderly_test_start()`, you must pass in the path to the report in question.

Usage

```
orderly_run_info(path = NULL)
```

Arguments

`path`	Path to the report currently being run. This should be left as `NULL` when running a report, and the path to the report being run should be used when using `orderly_test_start()`

Value

A list of metadata about the current report

Warning

It is important that this data is treated as *readonly*!

Examples

```
path <- orderly1::orderly_example("demo")

# This example uses orderly_run_info within its script, saving the
# output to "output.rds"
readLines(file.path(path, "src", "use_dependency", "script.R"))

# Run the dependency:
id <- orderly1::orderly_run("other", list(nmin = 0), root = path)
orderly1::orderly_commit(id, root = path)

# Then the report
id <- orderly1::orderly_run("use_dependency", root = path)

# This is the contents:
readRDS(file.path(path, "draft", "use_dependency", id, "info.rds"))
```

`orderly_run_remote` *Run a report on a remote server*

Description

Run a report on a remote server. Note that this is only supported for remotes using OrderlyWeb at present.

Usage

```
orderly_run_remote(
  name,
  parameters = NULL,
  ref = NULL,
  timeout = NULL,
  wait = 3600,
  poll = 1,
  open = TRUE,
  stop_on_error = TRUE,
  stop_on_timeout = TRUE,
  progress = TRUE,
  root = NULL,
  locate = TRUE,
  instance = NULL,
  remote = NULL
)
```

Arguments

`name`	Name of the report
`parameters`	Parameters for the report
`ref`	Optional reference, indicating which branch should be used. This cannot be used if the remote has `default_branch_only` set.
`timeout`	Time to tell the server to wait before killing the report.
`wait`	Time to wait for the report to be run; if the report takes longer than this time to run but `timeout` is longer it will remain running on the server but we will stop waiting for it and instead throw an error.
`poll`	Period to poll the server for results (in seconds)
`open`	Logical, indicating if the report should be opened in a browser on completion (if supported by the remote)
`stop_on_error`	Logical, indicating if we should throw an error if the report fails. If you set this to `FALSE` it will be much easier to debug, but more annoying in scripts. If the report times out on the server (i.e., takes longer than `timeout`) that counts as an error.

stop_on_timeout
: Logical, indicating if we should throw an error if the report takes longer than `wait` seconds to complete.

progress
: Logical, indicating if a progress spinner should be included.

root
: The path to an orderly root directory, or `NULL` (the default) to search for one from the current working directory if `locate` is `TRUE`.

locate
: Logical, indicating if the configuration should be searched for. If `TRUE` and `config` is not given, then orderly looks in the working directory and up through its parents until it finds an `orderly_config.yml` file.

instance
: Select instance of the source database to be used, where multiple instances are configured. Use a single unnamed character string to indicate an instance to match. Will use default if NULL.

remote
: Description of the location. Typically this is a character string indicating a remote specified in the `remotes` block of your `orderly_config.yml`. It is also possible to pass in a directly created remote object (e.g., using `orderly_remote_path()`, or one provided by another package). If left `NULL`, then the default remote for this orderly repository is used - by default that is the first listed remote.

Value

No return value, this function is called only for its side effects

Examples

```
path_remote <- orderly1::orderly_example("demo")
path_local <- orderly1::orderly_example("demo")
remote <- orderly1::orderly_remote_path(path_remote)
# Currently, path remotes don't support run
try(orderly1::orderly_run_remote(
  "minimal", remote = remote, root = path_local))
```

---

orderly_search *Search for orderly reports matching criteria*

Description

Search for orderly reports matching criteria. This can be used to find reports where a particular parameter or tag was used (it will likely be expanded as time goes on - let us know if that would be useful). We search within versions of a single report only.

Usage

```
orderly_search(
  query,
  name,
  parameters = NULL,
```

```
  draft = FALSE,
  root = NULL,
  locate = TRUE,
  remote = NULL
)
```

Arguments

query	The query string - see details and examples
name	Name of the report to search. Only a single report can be searched at once.
parameters	Named list of parameters (as would be passed to orderly_run()) if your query uses parameters on the right-hand-side of an expression.
draft	Should draft reports be used searched? This should be used only in development. Valid values are logical (TRUE, FALSE) or use the string newer to use draft reports where they are newer than archive reports. For consistency, always and never are equivalent to TRUE and FALSE, respectively.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
remote	A remote to use, if you want to apply the query remotely. If this is used then draft cannot be set to TRUE as remotes do not expose draft reports.

Details

The query syntax is deliberately very simple; it may expand a bit later. At this point you can search for parameters and for tags, and these can be combined. Note that if you are using OrderlyWeb, then only orderly (and not OrderlyWeb) tags are searched.

The idea here is that the queries can be used to find ids that match certain criteria for use as dependencies. This function lets you work out what would be resolved by the query, and using this query string in a depends: section will let you select a report that matches some criteria. For example, suppose that you have report A that takes a parameter "fruit" with values like "apple", "banana", and a report B that depends on A. You could then write:

```
depends:
  A:
    id: latest(parameter:fruit == "apple")
    uses:
      summary.csv: summary.csv
```

To get the summary.csv file out of the latest report A that was run with the "fruit" parameter set to "apple". If "B" itself takes parameters, you can use those parameters in these query expressions like

```
depends:
  A:
```

```
  id: latest(parameter:fruit == target_fruit)
  uses:
    summary.csv: summary.csv
```

(assuming that B takes a parameter target_fruit).

The syntax for tags is simpler, one uses tag:tagname to test for presence of a tag called "tagname".

Search queries can be joined by && and || and grouped using parentheses, these groups (or tags) can be negated with !, so a complicated query expression might look like:

```
(parameter:fruit == "apple" && !tag:weekly) || parameter:fruit == "banana"
```

Be careful of comparing floating point numbers with == or != as they may not always return what you expect (for example sqrt(3)^2 == 3 is FALSE).

In the documentation and error messages we may refer to the left-hand-side of : as a "namespace". At this point the only supported namespaces are tag and parameter.

Value

A character vector of matching report ids, possibly zero-length. If the query is a "latest" query, then exactly one report id, possibly NA.

Examples

```
# We need a few reports here to actually query.  There is a report in
# the "demo" example called "other" that takes a parameter "nmin",
# which is used to filter data - it's not terribly important what it
# does here, but it can give us a set of reports to use.

# The demo set also includes configuration for two tags, called
# "dataset" and "plot" - the "dataset" tag will always be applied
# as it is listed in the orderly.yml but we can still add the
# "plot" tag interactively
root <- orderly1::orderly_example("demo")

# A helper function to mass-produce reports will reduce noise a bit
run1 <- function(nmin, tags = NULL) {
  id <- orderly_run("other", root = root, echo = FALSE,
                    parameters = list(nmin = nmin), tags = tags)
  orderly_commit(id, root = root)
  id
}

ids <- c(run1(0.1), run1(0.2, "plot"), run1(0.3))

# We can then ask for all reports where the parameter nmin was more
# than some value
orderly1::orderly_search("parameter:nmin > 0.15", "other", root = root)

# Or use "&&" to find tags within a range
orderly1::orderly_search("parameter:nmin > 0.1 && parameter:nmin < 0.3",
```

```
                        "other", root = root)

# If a parameter is not present in some versions of a report you
# can use is.null to test for it (this is only ever the case if
# you have altered a report definition to add or remove a
# parameter)
orderly1::orderly_search("is.null(parameter:nmin)", "other", root = root)

# We can look for tags
orderly1::orderly_search("tag:plot", "other", root = root)

# or exclude them
orderly1::orderly_search("!tag:plot", "other", root = root)

# or combine that with the presence/absence of a tag
orderly1::orderly_search("parameter:nmin > 0.15 && !tag:plot",
                        "other", root = root)

# Use latest() over a query to find the latest report matching the
# query expression.
orderly1::orderly_search("latest(parameter:nmin > 0.15)",
                        "other", root = root)

# If no reports are found, then a zero-length character vector is returned
orderly1::orderly_search("parameter:nmin > 0.4", "other", root = root)

# Or, in the case of latest(), NA
orderly1::orderly_search("latest(parameter:nmin > 0.4)",
                        "other", root = root)
```

`orderly_test_start` *Prepare a directory for orderly to use*

Description

For interactive testing of orderly code. This runs through and sets everything up as orderly would (creates a new working directory and copies files into it, pulls data from the database, copies over any dependent reports) but then rather than running the report hands back to the user.

Usage

```
orderly_test_start(
  name,
  parameters = NULL,
  envir = parent.frame(),
  root = NULL,
  locate = TRUE,
  instance = NULL,
  use_draft = FALSE,
```

```
  remote = NULL
)

orderly_test_check(path = NULL)
```

Arguments

name	Name of the report to run (see orderly_list()). A leading src/ will be removed if provided, allowing easier use of autocomplete.
parameters	Parameters passed to the report. A named list of parameters declared in the orderly.yml. Each parameter must be a scalar character, numeric, integer or logical.
envir	The parent of the environment that will be used to evaluate the report script; by default a new environment will be made with the global environment as the parent.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
instance	Select instance of the source database to be used, where multiple instances are configured. Use a single *unnamed* character string to indicate an instance to match. If given, then this name must be present in all databases where instances are listed in orderly_config.yml, and will be ignored by all database where instances are not given. See the "orderly" vignette for further information.
use_draft	Should draft reports be used for dependencies? This should be used only in development. Valid values are logical (TRUE, FALSE) or use the string newer to use draft reports where they are newer than archive reports. For consistency, always and never are equivalent to TRUE and FALSE, respectively.
remote	Remote to use to resolve dependencies. Use this in order to run a report with the same dependencies as are available on a remote server, particularly when using id = "latest". Note that this is not the same as running orderly_pull_dependencies(), then orderly_run with remote = NULL, as the pull/run approach will use the latest report in *your* archive but the remote = "remote" approach will use the latest approach in the *remote* archive (which might be less recent).
path	Path to the report that is currently being run

Details

Previous versions of orderly changed into the created directory when using orderly1::orderly_test_start, which allowed interactive testing of a report, including ensuring that it has created all expected outputs. However, CRAN rules do not allow changing the working directory, which significantly reduces the usefulness of this function - as such we may remove it entirely in a future version of orderly if it does not prove useful in this more limited form.

The new suggested workflow is:

1. run orderly_test_start(...) to prepare a report directory

2. manually change into that directory following the printed instructions
3. use `orderly_test_check` to check that your report has created the expected artefacts
4. manually change back to your original directory

Value

The path to the report directory

Examples

```
path <- orderly1::orderly_example("minimal")
p <- orderly1::orderly_test_start("example", root = path)

# The data in the orderly example is now available to use
dat

# Check to see which artefacts have been created so far:
orderly1::orderly_test_check(p)

# Manually the code that this report has in its script
png(file.path(p, "mygraph.png"))
barplot(setNames(dat$number, dat$name), las = 2)
dev.off()

# We now confirm that the artefact has been created:
orderly1::orderly_test_check(p)
```

`orderly_use_resource` *Add a resource to orderly.yml*

Description

Add one or more resources to an `orderly.yml` file.

Usage

```
orderly_use_resource(
  resources,
  name = NULL,
  root = NULL,
  locate = TRUE,
  show = TRUE,
  edit = TRUE,
  prompt = TRUE
)

orderly_use_source(
  sources,
```

```
  name = NULL,
  root = NULL,
  locate = TRUE,
  show = TRUE,
  edit = TRUE,
  prompt = TRUE
)

orderly_use_package(
  packages,
  name = NULL,
  root = NULL,
  locate = TRUE,
  show = TRUE,
  edit = TRUE,
  prompt = TRUE
)

orderly_use_gitignore(
  root = NULL,
  locate = TRUE,
  show = TRUE,
  edit = TRUE,
  prompt = TRUE
)
```

Arguments

resources, sources	Character vector of resources or sources to add. These must be filenames relative to the report directory, must exist, and must not already be present in the orderly.yml
name	Name of the report to modify. Like orderly_develop_start() this can be NULL if you have already set the working directory to be the source directory.
root	The path to an orderly root directory, or NULL (the default) to search for one from the current working directory if locate is TRUE.
locate	Logical, indicating if the configuration should be searched for. If TRUE and config is not given, then orderly looks in the working directory and up through its parents until it finds an orderly_config.yml file.
show	Logical, indicating if we should print the proposed changes to screen
edit	Logical, indicating if we should actually edit the orderly.yml file.
prompt	Logical, indicating if we should prompt before editing the orderly.yml file. Only has an effect if edit is TRUE.
packages	Character vector of package names to add. These must not already exist in the orderly.yml

Details

The `orderly_use_gitignore` configures a basic `.gitignore` file at the root of your orderly project that will prevent files from being added to git. This is only really useful if you are using (or will use) git, but it is harmless at worst.

Value

Invisibly, this function returns information about the file it would edit. This information is primarily for debugging purposes and the format is subject to change.

Examples

```
path <- orderly1::orderly_example("minimal")

# Suppose we wanted to use the mtcars data within our report.
# First, the file must exist:
write.csv(mtcars, file.path(path, "src", "example", "mtcars.csv"),
          row.names = FALSE)

# Preview expected changes
orderly1::orderly_use_resource("mtcars.csv", "example", path, edit = FALSE)

# Modify the orderly.yml file within src/example:
orderly1::orderly_use_resource("mtcars.csv", "example", path,
                               prompt = FALSE)

# The result is a file that now has a 'resources' section
# containing our new file
writeLines(readLines(file.path(path, "src", "example", "orderly.yml")))

# (of course, we'd still need to modify the script to use it).
```

Index