Package 'orderly'

January 12, 2020

Title Lightweight Reproducible Reporting

Version 1.0.4

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://github.com/vimc/orderly

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

SystemRequirements git

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

Suggests httr, jsonlite, knitr, mockery, processx, rmarkdown, testthat, vaultr (>= 1.0.0)

RoxygenNote 6.1.1

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Author Rich FitzJohn [aut, cre],
Robert Ashton [aut],
Alex Hill [aut],
Martin Eden [aut],
Wes Hinsley [aut],
Emma Russell [aut],
James Thompson [aut],
Imperial College of Science, Technology and Medicine [cph]

Maintainer Rich FitzJohn <rich.fitzjohn@gmail.com>

Repository CRAN

Date/Publication 2020-01-12 14:40:02 UTC

R topics documented:

`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 <- orderly::orderly_example("minimal")
id1 <- orderly::orderly_run("example", root = path)
id2 <- orderly::orderly_run("example", root = path)
orderly::orderly_commit(id2, root = path)

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

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

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

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.

Value

The path to the newly committed report

Examples

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

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

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

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.

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

# The source database holds the data that might be accessible via
# the 'data' entry in orderly.yml:
db <- orderly::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 <- orderly::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 (see this Wikipedia page for more information). 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 <- orderly::orderly_example("demo")
id1 <- orderly::orderly_run("minimal", root = path)
id2 <- orderly::orderly_run("minimal", root = path)
orderly_commit(id1, root = path)
orderly_commit(id2, root = path)
tryCatch(
  orderly::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 <- orderly::orderly_example("demo")
id <- orderly::orderly_run("other", list(nmin = 0),
                           root = path_remote, echo = FALSE)
orderly::orderly_commit(id, root = path_remote)
id <- orderly::orderly_run("use_dependency",
```

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

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

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

# There is no remote set by default:
try(orderly::orderly_default_remote_get(root = path_local))

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

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

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

`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)
```

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.

Value

Returns the path to the orderly example

Examples

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

# Example reports within this repository:
orderly::orderly_list(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 <- orderly::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	Find most recent *draft* report
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 <- orderly::orderly_example("minimal")
id1 <- orderly::orderly_run("example", root = path, echo = FALSE)
id2 <- orderly::orderly_run("example", root = path, echo = FALSE)

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

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

# Commit the second report and it will be reported as latest instead:
orderly::orderly_commit(id2, root = path)
orderly::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 <- orderly::orderly_example("demo")

# Reports that _could_ be run:
orderly::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)

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.

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 <- orderly::orderly_example("demo")

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

# Run a report twice:
```

```
id1 <- orderly::orderly_run("minimal", root = path)
id2 <- orderly::orderly_run("minimal", root = path)

# We can see both drafts:
orderly::orderly_list_drafts(path)

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

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

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

# And the second report is in the archive:
orderly::orderly_list_archive(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 (!orderly::orderly_log_off()) {
  on.exit(orderly::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 <- orderly::orderly_log_on()

path <- orderly::orderly_example("minimal")

# By default we get both orderly log messages (e.g.,
# "[name] example") and the output of R when it runs the report:
orderly::orderly_run("example", root = path)

# Passing FALSE to the echo argument suppresses R's output but not
# orderly messages:
orderly::orderly_run("example", root = path, echo = FALSE)

# Disabling the log suppresses orderly's messages but still
# displays R's output:
orderly::orderly_log_off()
orderly::orderly_run("example", root = path)

# And using both will prevent all output
orderly::orderly_run("example", root = path, echo = FALSE)

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

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

# Restore to previous settings:
if (logging_was_enabled) {
  orderly::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 <- orderly::orderly_example("minimal")
orderly::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 `templates` in your project root. The special label "orderly" will use orderly's builtin template.

Details

To create a custom template, create a directory `templates` 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 <- orderly::orderly_example("minimal")

# Create a new report with the name "myreport" in this orderly
# repository:
orderly::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_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, root = NULL, locate = TRUE,
  remote = NULL)

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

Arguments

name	Name of the report to download dependencies for
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.
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.

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 = "orderly") 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 <- orderly::orderly_example("demo")
id <- orderly::orderly_run("other", list(nmin = 0),
                           root = path_remote, echo = FALSE)
orderly::orderly_commit(id, root = path_remote)
id <- orderly::orderly_run("use_dependency",
                           root = path_remote, echo = FALSE)
orderly::orderly_commit(id, root = path_remote)

# We'll create a an object to interact with this remote using
# orderly_remote_path.
remote <- orderly::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 <- orderly::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(orderly::orderly_run("use_dependency", root = path_local))

# We can "pull" depenencies of a report before running
orderly::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:
orderly::orderly_run("use_dependency", root = path_local)

# We can also directly pull previously run reports:
orderly::orderly_pull_archive("use_dependency", id, remote = remote,
                              root = path_local)
orderly::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 <- orderly::orderly_example("minimal")
id <- orderly::orderly_run("example", root = path)
orderly::orderly_commit(id, root = path)

con <- orderly::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"))
orderly::orderly_rebuild(path)
file.exists(file.path(path, "orderly.sqlite"))
con <- orderly::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.
orderly::orderly_rebuild(path)
```

`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 <- orderly::orderly_example("demo")
id <- orderly::orderly_run("other", list(nmin = 0),
                           root = path_remote, echo = FALSE)
orderly::orderly_commit(id, root = path_remote)
id <- orderly::orderly_run("use_dependency",
                           root = path_remote, echo = FALSE)
orderly::orderly_commit(id, root = path_remote)

# We'll create a an object to interact with this remote using
# orderly_remote_path.
remote <- orderly::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 <- orderly::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(orderly::orderly_run("use_dependency", root = path_local))

# We can "pull" depenencies of a report before running
orderly::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:
orderly::orderly_run("use_dependency", root = path_local)

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

`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, parameters = NULL, envir = NULL, root = NULL,
  locate = TRUE, echo = TRUE, id_file = NULL, fetch = FALSE,
  ref = NULL, message = NULL)
```

Arguments

`name`	Name of the report to run (see `orderly_list`).
`parameters`	Parameters passed to the report. A named list of parameters declared in the `orderly.yml`.
`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
id_file	Write the identifier into a file
fetch	Logical, indicating if git should be fetched before checking out the reference ref.
ref	A git reference to use for this run (see Details)
message	An optional character string containing a message explaining why the report was run

Details

If ref is provided then before running a report orderly will try to check out (as a detached HEAD) ref, interpreted as a git reference. This can be a commit, tag, or a branch name (including remote). The working directory must be clean according to git status and this *will* require some careful use of .gitignore to exclude draft, archive, data and orderly.sqlite. The git tree will revert back to the original branch at completion (or failure to complete) the report.

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

```
id <-orderly::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.

Value

The id of the newly created report

See Also

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

Examples

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

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

`orderly_runner` *Orderly runner*

Description

An orderly runner. This is used to run reports as a server process. It's designed to be used in conjunction with OrderlyWeb, so there is no "draft" stage and reports are committed as soon as they are run. This function is not intended for human end users, only for creating automated tools for use with orderly.

Usage

```
orderly_runner(path, allow_ref = NULL, backup_period = 600)
```

Arguments

`path`	Path to use
`allow_ref`	Allow git to change branches/ref for run. If not given, then we will look to see if the orderly configuration disallows branch changes (based on the `ORDERLY_API_SERVER_IDENTITY` environment variable and the `master_only` setting of the relevant server block.
`backup_period`	Period (in seconds) between DB backups. This is a guide only as backups cannot happen while a task is running - if more than this many seconds have elapsed when the runner is in its idle loop a backup of the db will be performed. This creates a copy of orderly's destination database in `backup/db` with the same filename as the destination database, even if that database typically lives outside of the orderly tree. In case of corruption of the database, this backup can be manually moved into place. This is only needed if you are storing information alongside the core orderly tables (as done by OrderlyWeb).

Value

A runner object, with methods designed for internal use only.

Examples

```
path <- orderly::orderly_example("demo")
runner <- orderly::orderly_runner(path)
```

`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

Examples

```
path <- orderly::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 <- orderly::orderly_run("other", list(nmin = 0), root = path)
orderly::orderly_commit(id, root = path)

# Then the report
id <- orderly::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, 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 `master_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.
`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 <- orderly::orderly_example("demo")
path_local <- orderly::orderly_example("demo")
remote <- orderly::orderly_remote_path(path_remote)
# Currently, path remotes don't support run
try(orderly::orderly_run_remote(
  "minimal", remote = remote, root = path_local))
```

`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. The `orderly_data` function returns an environment with the extracted data.

Usage

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

orderly_test_check(path = NULL)

orderly_data(name, parameters = NULL, envir = NULL, root = NULL,
  locate = TRUE)
```

Arguments

`name`	Name of the report to run (see `orderly_list`).
`parameters`	Parameters passed to the report. A named list of parameters declared in the `orderly.yml`.
`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.
`path`	Path to the report that is currently being run

Details

Previous versions of orderly changed into the created directory when using `orderly::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 <- orderly::orderly_example("minimal")
p <- orderly::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:
orderly::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:
orderly::orderly_test_check(p)
# The function orderly_data does all the preparation work that
# orderly_run does, but does not run the report; instead it
# returns the created environment with all the data and parameters
# set.
path <- orderly::orderly_example("demo")
env <- orderly::orderly_data("other", list(nmin = 0.2), root = path)
ls(env)
env$nmin
env$extract
```

Index