Package 'tarchetypes'

April 17, 2024

Title Archetypes for Targets

Description Function-oriented Make-like declarative pipelines for
Statistics and data science are supported in the 'targets' R package.
As an extension to 'targets', the 'tarchetypes' package provides
convenient user-side functions to make 'targets' easier to use.
By establishing reusable archetypes for common kinds of
targets and pipelines, these functions help express complicated
reproducible pipelines concisely and compactly.
The methods in this package were influenced by the 'drake' R package
by Will Landau (2018) <doi:10.21105/joss.00550>.

Version 0.9.0

License MIT + file LICENSE

URL https://docs.ropensci.org/tarchetypes/,
https://github.com/ropensci/tarchetypes

BugReports https://github.com/ropensci/tarchetypes/issues

Depends R (>= 3.5.0)

Imports dplyr (>= 1.0.0), fs (>= 1.4.2), parallel, rlang (>= 0.4.7),
secretbase (>= 0.4.0), targets (>= 1.6.0), tibble (>= 3.0.1),
tidyselect (>= 1.1.0), utils, vctrs (>= 0.3.4), withr (>=
2.1.2)

Suggests curl (>= 4.3), knitr (>= 1.28), quarto (>= 1.4), rmarkdown
(>= 2.1), testthat (>= 3.0.0), xml2 (>= 1.3.2)

Encoding UTF-8

Language en-US

Config/testthat/edition 3

RoxygenNote 7.3.1

NeedsCompilation no

Author William Michael Landau [aut, cre]
(<https://orcid.org/0000-0003-1878-3253>),
Samantha Oliver [rev] (<https://orcid.org/0000-0001-5668-1165>),
Tristan Mahr [rev] (<https://orcid.org/0000-0002-8890-5116>),
Eli Lilly and Company [cph]

Maintainer William Michael Landau <will.landau.oss@gmail.com>

Repository CRAN

Date/Publication 2024-04-17 17:50:02 UTC

R **topics documented:**

tarchetypes-package   *targets: Archetypes for Targets*

Description

A pipeline toolkit for R, the `targets` package brings together function-oriented programming and Make-like declarative pipelines for Statistics and data science. The `tarchetypes` package provides convenient helper functions to create specialized targets, making pipelines in targets easier and cleaner to write and understand.

tar_age   *Create a target that runs when the last run gets old*

Description

`tar_age()` creates a target that reruns itself when it gets old enough. In other words, the target reruns periodically at regular intervals of time.

Usage

```
tar_age(
  name,
  command,
  age,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
```

```
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Character of length 1, name of the target.
command	R code to run the target and return a value.
age	A difftime object of length 1, such as as.difftime(3, units = "days"). If the target's output data files are older than age (according to the most recent time stamp over all the target's output files) then the target will rerun. On the other hand, if at least one data file is younger than Sys.time() - age, then the ordinary invalidation rules apply, and the target may or not rerun. If you want to force the target to run every 3 days, for example, set age = as.difftime(3, units = "days").
pattern	Language to define branching for a target. For example, in a pipeline with numeric vector targets x and y, tar_target(z, x + y, pattern = map(x, y)) implicitly defines branches of z that each compute x[1] + y[1], x[2] + y[2], and so on. See the user manual for details.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Logical, whether to rerun the target if the user-specified storage format changed. The storage format is user-specified through tar_target() or tar_option_set().
repository	Logical, whether to rerun the target if the user-specified storage repository changed. The storage repository is user-specified through tar_target() or tar_option_set().
iteration	Logical, whether to rerun the target if the user-specified iteration method changed. The iteration method is user-specified through tar_target() or tar_option_set().
error	Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.

- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                         Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.
  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically

try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   A `targets::tar_cue()` object. (See the "Cue objects" section for background.) This cue object should contain any optional secondary invalidation rules, anything except the `mode` argument. `mode` will be automatically determined by the `age` argument of `tar_age()`.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

`tar_age()` uses the cue from `tar_cue_age()`, which uses the time stamps from `targets::tar_meta()$time`. See the help file of `targets::tar_timestamp()` for an explanation of how this time stamp is calculated.

Value

A target object. See the "Target objects" section for background.

Dynamic branches at regular time intervals

Time stamps are not recorded for whole dynamic targets, so `tar_age()` is not a good fit for dynamic branching. To invalidate dynamic branches at regular intervals, it is recommended to use `targets::tar_older()` in combination with `targets::tar_invalidate()` right before calling `tar_make()`. For example, `tar_invalidate(any_of(tar_older(Sys.time - as.difftime(1, units = "weeks"))))` # nolint invalidates all targets more than a week old. Then, the next `tar_make()` will rerun those targets.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other cues: tar_cue_age(), tar_cue_age_raw(), tar_cue_force(), tar_cue_skip()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  library(tarchetypes)
  list(
    tarchetypes::tar_age(
      data,
      data.frame(x = seq_len(26)),
      age = as.difftime(0.5, units = "secs")
    )
  )
})
targets::tar_make()
Sys.sleep(0.6)
targets::tar_make()
})
}
```

---

tar_change   *Target that responds to an arbitrary change.*

Description

Create a target that responds to a change in an arbitrary value. If the value changes, the target reruns.

Usage

```
tar_change(
  name,
  command,
  change,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
```

```
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.
command	R code to run the target.
change	R code for the upstream change-inducing target.
tidy_eval	Whether to invoke tidy evaluation (e.g. the `!!` operator from `rlang`) as soon as the target is defined (before `tar_make()`). Applies to arguments `command` and `change`.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

- "gcp": Google Cloud Platform storage bucket. See the cloud storage sec-
  tion of https://books.ropensci.org/targets/data.html for details for
  instructions.

Note: if repository is not "local" and format is "file" then the target
should create a single output file. That output file is uploaded to the cloud and
tracked for changes where it exists in the cloud. The local file is deleted after
the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation
  happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of
  a non-dynamic data frame. For iteration = "group", the target must not
  by dynamic (the pattern argument of tar_target() must be left NULL).
  The target's return value must be a data frame with a special tar_group
  column of consecutive integers from 1 through the number of groups. Each
  integer designates a group, and a branch is created for each collection of
  rows in a group. See the tar_group() function to see how you can create
  the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new tar-
  gets launch after that. (Visit https://books.ropensci.org/targets/
  debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is
  deliberately wrong so the target is not up to date for the next run of the
  pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                  memory until the end of the pipeline (unless storage is "worker", in which case
                  targets unloads the value from memory right after storing it in order to avoid
                  sending copious data over a network). If "transient", the target gets unloaded
                  after every new target completes. Either way, the target gets automatically
                  loaded into memory whenever another target needs the value. For cloud-based
                  dynamic files (e.g. format = "file" with repository = "aws"), this memory
                  strategy applies to the temporary local copy of the file: "persistent" means
                  it remains until the end of the pipeline and is then deleted, and "transient"
                  means it gets deleted as soon as possible. The former conserves bandwidth, and
                  the latter conserves local storage.

garbage_collection
                  Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                  central controlling R process. Otherwise, if deployment is "worker" and you
                  set up the pipeline with distributed/parallel computing, then the target runs on a
                  parallel worker. For more on distributed/parallel computing in targets, please
                  visit https://books.ropensci.org/targets/crew.html.

priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval	Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date. Only applies to the downstream target. The upstream target always runs.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_change() creates a pair of targets, one upstream and one downstream. The upstream target always runs and returns an auxiliary value. This auxiliary value gets referenced in the downstream

target, which causes the downstream target to rerun if the auxiliary value changes. The behavior is cancelled if cue is tar_cue(depend = FALSE) or tar_cue(mode = "never").

Because the upstream target always runs, tar_outdated() and tar_visnetwork() will always show both targets as outdated. However, tar_make() will still skip the downstream one if the upstream target did not detect a change.

Value

A list of two target objects, one upstream and one downstream. The upstream one triggers the change, and the downstream one responds to it. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other targets with custom invalidation rules: tar_download(), tar_force(), tar_skip()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_change(x, command = tempfile(), change = tempfile())
  )
})
targets::tar_make()
targets::tar_make()
})
}
```

---

tar_combine   *Static aggregation.*

Description

Aggregate the results of upstream targets into a new target.

Usage

```
tar_combine(
  name,
  ...,
  command = vctrs::vec_c(!!!.x),
  use_names = TRUE,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the new target.
...	One or more target objects or list of target objects. Lists can be arbitrarily nested, as in `list()`.
command	R command to aggregate the targets. Must contain `!!!.x` where the arguments are to be inserted, where `!!!` is the unquote splice operator from `rlang`.
use_names	Logical, whether to insert the names of the targets into the command when splicing.
pattern	Language to define branching for a target. For example, in a pipeline with numeric vector targets x and y, `tar_target(z, x + y, pattern = map(x, y))` implicitly defines branches of z that each compute `x[1] + y[1]`, `x[2] + y[2]`, and so on. See the user manual for details.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of `tar_target()` must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the `tar_group()` function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

                    Logical, whether to run `base::gc()` just before the target runs.

deployment   Character of length 1. If `deployment` is `"main"`, then the target will run on the central controlling R process. Otherwise, if `deployment` is `"worker"` and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in `targets`, please visit <https://books.ropensci.org/targets/crew.html>.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [`tar_make_future()`](#)).

resources   Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.

storage   Character of length 1, only relevant to [`tar_make_clustermq()`](#) and [`tar_make_future()`](#). Must be one of the following values:

- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"worker"`: the worker saves/uploads the value.
- `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

                    If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

                    The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to [`tar_make_clustermq()`](#) and [`tar_make_future()`](#). Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [`tar_manifest()`](#) and [`tar_visnetwork()`](#), and they let you select subsets of targets for the `names` argument of functions like [`tar_make()`](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Value

A new target object to combine the return values from the upstream targets. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other branching: `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`, `tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size()`, `tar_map2_size_raw()`, `tar_map_rep()`, `tar_map_rep_raw()`, `tar_rep()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  target1 <- targets::tar_target(x, head(mtcars))
  target2 <- targets::tar_target(y, tail(mtcars))
  target3 <- tarchetypes::tar_combine(
    new_target_name,
    target1,
    target2,
    command = bind_rows(!!!.x)
  )
  list(target1, target2, target3)
})
targets::tar_manifest()
})
}
```

---

tar_combine_raw   *Static aggregation (raw version).*

Description

Like `tar_combine()` except the name, command, and `pattern` arguments use standard evaluation.

Usage

```
tar_combine_raw(
  name,
  ...,
  command = expression(vctrs::vec_c(!!!.x)),
  use_names = TRUE,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Character, name of the new target.
...	One or more target objects or list of target objects. Lists can be arbitrarily nested, as in `list()`.
command	Expression object, R command to aggregate the targets. Must contain `!!!.x` where the arguments are to be inserted, where `!!!` is the unquote splice operator from `rlang`.
use_names	Logical, whether to insert the names of the targets into the command when splicing.
pattern	Similar to the `pattern` argument of [tar_target()](#) except the object must already be an expression instead of informally quoted code. `base::expression()` and `base::quote()` can produce such objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of `tar_target()` must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the `tar_group()` function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                      central controlling R process. Otherwise, if deployment is "worker" and you
                      set up the pipeline with distributed/parallel computing, then the target runs on a
                      parallel worker. For more on distributed/parallel computing in targets, please
                      visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                      when multiple competing targets are ready simultaneously. Targets with priori-
                      ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                      computing functionality, alternative data storage formats, and other optional ca-
                      pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                      Must be one of the following values:

                      • "main": the target's return value is sent back to the host machine and
                        saved/uploaded locally.
                      • "worker": the worker saves/uploads the value.
                      • "none": almost never recommended. It is only for niche situations, e.g. the
                        data needs to be loaded explicitly from another language. If you do use it,
                        then the return value of the target is totally ignored when the target ends,
                        but each downstream target still attempts to load the data file (except when
                        retrieval = "none").
                        If you select storage = "none", then the return value of the target's com-
                        mand is ignored, and the data is not saved automatically. As with dynamic
                        files (format = "file") it is the responsibility of the user to write to the
                        data store from inside the target.
                        The distinguishing feature of storage = "none" (as opposed to format =
                        "file") is that in the general case, downstream targets will automatically
                        try to load the data from the data store as a dependency. As a corollary,
                        storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                      Must be one of the following values:

                      • "main": the target's dependencies are loaded on the host machine and sent
                        to the worker before the target runs.
                      • "worker": the worker loads the targets dependencies.
                      • "none": the dependencies are not loaded at all. This choice is almost never
                        recommended. It is only for niche situations, e.g. the data needs to be
                        loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether
                      the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                      target. Descriptions appear as target labels in functions like tar_manifest()
                      and tar_visnetwork(), and they let you select subsets of targets for the names
                      argument of functions like tar_make(). For example, tar_manifest(names =
                      tar_described_as(starts_with("survival model"))) lists all the targets
                      whose descriptions start with the character string "survival model".

Value

A new target object to combine the return values from the upstream targets. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at <https://books.ropensci.org/targets/>. Please read the walkthrough at <https://books.ropensci.org/targets/walkthrough.html> to understand the role of target objects in analysis pipelines.

For developers, <https://wlandau.github.io/targetopia/contributing.html#target-factories> explains target factories (functions like this one which generate targets) and the design specification at <https://books.ropensci.org/targets-design/> details the structure and composition of target objects.

See Also

Other branching: tar_combine(), tar_map(), tar_map2(), tar_map2_count(), tar_map2_count_raw(), tar_map2_raw(), tar_map2_size(), tar_map2_size_raw(), tar_map_rep(), tar_map_rep_raw(), tar_rep(), tar_rep2(), tar_rep2_raw(), tar_rep_map(), tar_rep_map_raw(), tar_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  target1 <- targets::tar_target(x, head(mtcars))
  target2 <- targets::tar_target(y, tail(mtcars))
  target3 <- tarchetypes::tar_combine(new_target_name, target1, target2)
  list(target1, target2, target3)
})
targets::tar_manifest()
})
}
```

---

tar_cue_age   *Cue to run a target when the last output reaches a certain age*

Description

tar_cue_age() creates a cue object to rerun a target if the most recent output data becomes old enough. The age of the target is determined by targets::tar_timestamp(), and the way the time stamp is calculated is explained in the Details section of the help file of that function.

Usage

```
tar_cue_age(
  name,
  age,
  command = TRUE,
  depend = TRUE,
  format = TRUE,
  repository = TRUE,
  iteration = TRUE,
  file = TRUE
)
```

Arguments

name	Symbol, name of the target.
age	A `difftime` object of length 1, such as `as.difftime(3, units = "days")`. If the target's output data files are older than `age` (according to the most recent time stamp over all the target's output files) then the target will rerun. On the other hand, if at least one data file is younger than `Sys.time() - age`, then the ordinary invalidation rules apply, and the target may or not rerun. If you want to force the target to run every 3 days, for example, set `age = as.difftime(3, units = "days")`.
command	Logical, whether to rerun the target if command changed since last time.
depend	Logical, whether to rerun the target if the value of one of the dependencies changed.
format	Logical, whether to rerun the target if the user-specified storage format changed. The storage format is user-specified through [tar_target()](#) or [tar_option_set()](#).
repository	Logical, whether to rerun the target if the user-specified storage repository changed. The storage repository is user-specified through [tar_target()](#) or [tar_option_set()](#).
iteration	Logical, whether to rerun the target if the user-specified iteration method changed. The iteration method is user-specified through [tar_target()](#) or [tar_option_set()](#).
file	Logical, whether to rerun the target if the file(s) with the return value changed or at least one is missing.

Details

`tar_cue_age()` uses the time stamps from `tar_meta()$time`. If no time stamp is recorded, the cue defaults to the ordinary invalidation rules (i.e. `mode = "thorough"` in `targets::tar_cue()`).

Value

A cue object. See the "Cue objects" section for background.

Dynamic branches at regular time intervals

Time stamps are not recorded for whole dynamic targets, so `tar_age()` is not a good fit for dynamic branching. To invalidate dynamic branches at regular intervals, it is recommended to use

targets::tar_older() in combination with targets::tar_invalidate() right before calling tar_make(). For example, tar_invalidate(any_of(tar_older(Sys.time - as.difftime(1, units = "weeks")))) # nolint invalidates all targets more than a week old. Then, the next tar_make() will rerun those targets.

Cue objects

A cue object is an object generated by targets::tar_cue(), tarchetypes::tar_cue_force(), or similar. It is a collection of decision rules that decide when a target is invalidated/outdated (e.g. when tar_make() or similar reruns the target). You can supply these cue objects to the tar_target() function or similar. For example, tar_target(x, run_stuff(), cue = tar_cue(mode = "always")) is a target that always calls run_stuff() during tar_make() and always shows as invalidated/outdated in tar_outdated(), tar_visnetwork(), and similar functions.

See Also

Other cues: [tar_age()](), [tar_cue_age_raw]()(), [tar_cue_force]()(), [tar_cue_skip]()()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  library(tarchetypes)
  list(
    targets::tar_target(
      data,
      data.frame(x = seq_len(26)),
      cue = tarchetypes::tar_cue_age(
        name = data,
        age = as.difftime(0.5, units = "secs")
      )
    )
  )
})
targets::tar_make()
Sys.sleep(0.6)
targets::tar_make()
})
}
```

---

tar_cue_age_raw   *Cue to run a target when the last run reaches a certain age (raw version)*

Description

tar_cue_age_raw() acts like tar_cue_age() except the name argument is a character string, not
a symbol. tar_cue_age_raw() creates a cue object to rerun a target if the most recent output data
becomes old enough. The age of the target is determined by targets::tar_timestamp(), and the
way the time stamp is calculated is explained in the Details section of the help file of that function.

Usage

```
tar_cue_age_raw(
  name,
  age,
  command = TRUE,
  depend = TRUE,
  format = TRUE,
  repository = TRUE,
  iteration = TRUE,
  file = TRUE
)
```

Arguments

name	Character of length 1, name of the target.
age	A difftime object of length 1, such as as.difftime(3, units = "days"). If the target's output data files are older than age (according to the most recent time stamp over all the target's output files) then the target will rerun. On the other hand, if at least one data file is younger than Sys.time() - age, then the ordinary invalidation rules apply, and the target may or not rerun. If you want to force the target to run every 3 days, for example, set age = as.difftime(3, units = "days").
command	Logical, whether to rerun the target if command changed since last time.
depend	Logical, whether to rerun the target if the value of one of the dependencies changed.
format	Logical, whether to rerun the target if the user-specified storage format changed. The storage format is user-specified through [tar_target()](#) or [tar_option_set()](#).
repository	Logical, whether to rerun the target if the user-specified storage repository changed. The storage repository is user-specified through [tar_target()](#) or [tar_option_set()](#).
iteration	Logical, whether to rerun the target if the user-specified iteration method changed. The iteration method is user-specified through [tar_target()](#) or [tar_option_set()](#).
file	Logical, whether to rerun the target if the file(s) with the return value changed or at least one is missing.

Details

tar_cue_age_raw() uses the time stamps from tar_meta()$time. If no time stamp is recorded,
the cue defaults to the ordinary invalidation rules (i.e. mode = "thorough" in targets::tar_cue()).

Value

A cue object. See the "Cue objects" section for background.

Dynamic branches at regular time intervals

Time stamps are not recorded for whole dynamic targets, so `tar_age()` is not a good fit for dynamic branching. To invalidate dynamic branches at regular intervals, it is recommended to use `targets::tar_older()` in combination with `targets::tar_invalidate()` right before calling `tar_make()`. For example, `tar_invalidate(any_of(tar_older(Sys.time - as.difftime(1,` `units = "weeks"))))` # nolint invalidates all targets more than a week old. Then, the next `tar_make()` will rerun those targets.

Cue objects

A cue object is an object generated by `targets::tar_cue()`, `tarchetypes::tar_cue_force()`, or similar. It is a collection of decision rules that decide when a target is invalidated/outdated (e.g. when `tar_make()` or similar reruns the target). You can supply these cue objects to the `tar_target()` function or similar. For example, `tar_target(x, run_stuff(), cue = tar_cue(mode` `= "always"))` is a target that always calls `run_stuff()` during `tar_make()` and always shows as invalidated/outdated in `tar_outdated()`, `tar_visnetwork()`, and similar functions.

See Also

Other cues: [tar_age()](), [tar_cue_age()](), [tar_cue_force()](), [tar_cue_skip()]()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  library(tarchetypes)
  list(
    targets::tar_target(
      data,
      data.frame(x = seq_len(26)),
      cue = tarchetypes::tar_cue_age_raw(
        name = "data",
        age = as.difftime(0.5, units = "secs")
      )
    )
  )
})
targets::tar_make()
Sys.sleep(0.6)
targets::tar_make()
})
}
```

---

tar_cue_force   *Cue to force a target to run if a condition is true*

Description

tar_cue_force() creates a cue object to force a target to run if an arbitrary condition evaluates to TRUE. Supply the returned cue object to the cue argument of targets::tar_target() or similar.

Usage

```
tar_cue_force(
  condition,
  command = TRUE,
  depend = TRUE,
  format = TRUE,
  repository = TRUE,
  iteration = TRUE,
  file = TRUE
)
```

Arguments

condition	Logical vector evaluated locally when the target is defined. If any element of condition is TRUE, the target will definitely rerun when the pipeline runs. Otherwise, the target may or may not rerun, depending on the other invalidation rules. condition is evaluated when this cue factory is called, so the condition cannot depend on upstream targets, and it should be quick to calculate.
command	Logical, whether to rerun the target if command changed since last time.
depend	Logical, whether to rerun the target if the value of one of the dependencies changed.
format	Logical, whether to rerun the target if the user-specified storage format changed. The storage format is user-specified through [tar_target()](#) or [tar_option_set()](#).
repository	Logical, whether to rerun the target if the user-specified storage repository changed. The storage repository is user-specified through [tar_target()](#) or [tar_option_set()](#).
iteration	Logical, whether to rerun the target if the user-specified iteration method changed. The iteration method is user-specified through [tar_target()](#) or [tar_option_set()](#).
file	Logical, whether to rerun the target if the file(s) with the return value changed or at least one is missing.

Details

tar_cue_force() and [tar_force()](#) operate differently. The former defines a cue object based on an eagerly evaluated condition, and [tar_force()](#) puts the condition in a special upstream target that always runs. Unlike tar_cue_force(), the condition in [tar_force()](#) can depend on upstream targets, but the drawback is that targets defined with [tar_force()](#) will always show up as outdated in functions like tar_outdated() and tar_visnetwork() even though tar_make() may still skip the main target if the condition is not met.

Value

A cue object. See the "Cue objects" section for background.

Cue objects

A cue object is an object generated by `targets::tar_cue()`, `tarchetypes::tar_cue_force()`, or similar. It is a collection of decision rules that decide when a target is invalidated/outdated (e.g. when `tar_make()` or similar reruns the target). You can supply these cue objects to the `tar_target()` function or similar. For example, `tar_target(x, run_stuff(), cue = tar_cue(mode = "always"))` is a target that always calls `run_stuff()` during `tar_make()` and always shows as invalidated/outdated in `tar_outdated()`, `tar_visnetwork()`, and similar functions.

See Also

Other cues: `tar_age()`, `tar_cue_age()`, `tar_cue_age_raw()`, `tar_cue_skip()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  library(tarchetypes)
  list(
    targets::tar_target(
      data,
      data.frame(x = seq_len(26)),
      cue = tarchetypes::tar_cue_force(1 > 0)
    )
  )
})
targets::tar_make()
targets::tar_make()
})
}
```

---

tar_cue_skip   *Cue to skip a target if a condition is true*

Description

`tar_cue_skip()` creates a cue object to skip a target if an arbitrary condition evaluates to TRUE. The target still builds if it was never built before. Supply the returned cue object to the cue argument of `targets::tar_target()` or similar.

Usage

```
tar_cue_skip(
  condition,
  command = TRUE,
  depend = TRUE,
  format = TRUE,
  repository = TRUE,
  iteration = TRUE,
  file = TRUE
)
```

Arguments

condition	Logical vector evaluated locally when the target is defined. If any element of `condition` is TRUE, the pipeline will skip the target unless the target has never been built before. If all elements of `condition` are FALSE, then the target may or may not rerun, depending on the other invalidation rules. `condition` is evaluated when this cue factory is called, so the condition cannot depend on upstream targets, and it should be quick to calculate.
command	Logical, whether to rerun the target if command changed since last time.
depend	Logical, whether to rerun the target if the value of one of the dependencies changed.
format	Logical, whether to rerun the target if the user-specified storage format changed. The storage format is user-specified through [tar_target()](#) or [tar_option_set()](#).
repository	Logical, whether to rerun the target if the user-specified storage repository changed. The storage repository is user-specified through [tar_target()](#) or [tar_option_set()](#).
iteration	Logical, whether to rerun the target if the user-specified iteration method changed. The iteration method is user-specified through [tar_target()](#) or [tar_option_set()](#).
file	Logical, whether to rerun the target if the file(s) with the return value changed or at least one is missing.

Value

A cue object. See the "Cue objects" section for background.

Cue objects

A cue object is an object generated by `targets::tar_cue()`, `tarchetypes::tar_cue_force()`, or similar. It is a collection of decision rules that decide when a target is invalidated/outdated (e.g. when `tar_make()` or similar reruns the target). You can supply these cue objects to the `tar_target()` function or similar. For example, `tar_target(x, run_stuff(), cue = tar_cue(mode = "always"))` is a target that always calls `run_stuff()` during `tar_make()` and always shows as invalidated/outdated in `tar_outdated()`, `tar_visnetwork()`, and similar functions.

See Also

Other cues: [tar_age()](#), [tar_cue_age()](#), [tar_cue_age_raw()](#), [tar_cue_force()](#)

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  library(tarchetypes)
  list(
    targets::tar_target(
      data,
      data.frame(x = seq_len(26)),
      cue = tarchetypes::tar_cue_skip(1 > 0)
    )
  )
})
targets::tar_make()
targets::tar_script({
  library(tarchetypes)
  list(
    targets::tar_target(
      data,
      data.frame(x = seq_len(25)), # Change the command.
      cue = tarchetypes::tar_cue_skip(1 > 0)
    )
  )
})
targets::tar_make()
targets::tar_make()
})
}
```

---

tar_download   *Target that downloads URLs.*

Description

Create a target that downloads file from one or more URLs and automatically reruns when the remote data changes (according to the ETags or last-modified time stamps).

Usage

```
tar_download(
  name,
  urls,
  paths,
  method = NULL,
  quiet = TRUE,
  mode = "w",
  cacheOK = TRUE,
  extra = NULL,
  headers = NULL,
```

```
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
urls	Character vector of URLs to track and download. Must be known and declared before the pipeline runs.
paths	Character vector of local file paths to download each of the URLs. Must be known and declared before the pipeline runs.
method	Method to be used for downloading files. Current download methods are "internal", "libcurl", "wget", "curl" and "wininet" (Windows only), and there is a value "auto": see 'Details' and 'Note'.
	The method can also be set through the option "download.file.method": see [options](#)().
quiet	If TRUE, suppress status messages (if any), and the progress bar.
mode	character. The mode with which to write the file. Useful values are "w", "wb" (binary), "a" (append) and "ab". Not used for methods "wget" and "curl". See also 'Details', notably about using "wb" for Windows.
cacheOK	logical. Is a server-side cached value acceptable?
extra	character vector of additional command-line arguments for the "wget" and "curl" methods.
headers	named character vector of additional HTTP headers to use in HTTP[S] requests. It is ignored for non-HTTP[S] URLs. The User-Agent header taken from the HTTPUserAgent option (see [options](#)) is automatically used as the first header.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of [tar_target()](#) must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [tar_group()](#) function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                   Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                   Must be one of the following values:

                   • "main": the target's return value is sent back to the host machine and
                     saved/uploaded locally.
                   • "worker": the worker saves/uploads the value.
                   • "none": almost never recommended. It is only for niche situations, e.g. the
                     data needs to be loaded explicitly from another language. If you do use it,
                     then the return value of the target is totally ignored when the target ends,
                     but each downstream target still attempts to load the data file (except when
                     retrieval = "none").
                     If you select storage = "none", then the return value of the target's com-
                     mand is ignored, and the data is not saved automatically. As with dynamic
                     files (format = "file") it is the responsibility of the user to write to the
                     data store from inside the target.
                     The distinguishing feature of storage = "none" (as opposed to format =
                     "file") is that in the general case, downstream targets will automatically
                     try to load the data from the data store as a dependency. As a corollary,
                     storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                   Must be one of the following values:

                   • "main": the target's dependencies are loaded on the host machine and sent
                     to the worker before the target runs.
                   • "worker": the worker loads the targets dependencies.
                   • "none": the dependencies are not loaded at all. This choice is almost never
                     recommended. It is only for niche situations, e.g. the data needs to be
                     loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether
                   the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                   target. Descriptions appear as target labels in functions like tar_manifest()
                   and tar_visnetwork(), and they let you select subsets of targets for the names
                   argument of functions like tar_make(). For example, tar_manifest(names =
                   tar_described_as(starts_with("survival model"))) lists all the targets
                   whose descriptions start with the character string "survival model".

Details

tar_download() creates a pair of targets, one upstream and one downstream. The upstream target
uses format = "url" (see targets::tar_target()) to track files at one or more URLs, and auto-
matically invalidate the target if the ETags or last-modified time stamps change. The downstream
target depends on the upstream one, downloads the files, and tracks them using format = "file".

Value

A list of two target objects, one upstream and one downstream. The upstream one watches a URL
for changes, and the downstream one downloads it. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other targets with custom invalidation rules: `tar_change()`, `tar_force()`, `tar_skip()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_download(
      x,
      urls = c("https://httpbin.org/etag/test", "https://r-project.org"),
      paths = c("downloaded_file_1", "downloaded_file_2")
    )
  )
})
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_eval   *Evaluate multiple expressions created with symbol substitution.*

Description

Loop over a grid of values, create an expression object from each one, and then evaluate that expression. Helps with general metaprogramming.

Usage

```
tar_eval(expr, values, envir = parent.frame())
```

Arguments

expr	Starting expression. Values are iteratively substituted in place of symbols in expr to create each new expression, and then each new expression is evaluated.
values	List of values to substitute into expr to create the expressions. All elements of values must have the same length.
envir	Environment in which to evaluate the new expressions.

Value

A list of return values from the generated expression objects. Often, these values are target objects. See the "Target objects" section for background on target objects specifically.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Metaprogramming utilities: tar_eval_raw(), tar_sub(), tar_sub_raw()

Examples

```
# tar_map() is incompatible with tar_render() because the latter
# operates on preexisting tar_target() objects. By contrast,
# tar_eval() and tar_sub() iterate over the literal code
# farther upstream.
values <- list(
  name = lapply(c("name1", "name2"), as.symbol),
  file = list("file1.Rmd", "file2.Rmd")
)
tar_sub(list(name, file), values = values)
tar_sub(tar_render(name, file), values = values)
path <- tempfile()
file.create(path)
str(tar_eval(tar_render(name, path), values = values))
# So in your _targets.R file, you can define a pipeline like as below.
# Just make sure to set a unique name for each target
# (which tar_map() does automatically).
values <- list(
  name = lapply(c("name1", "name2"), as.symbol),
  file = c(path, path)
)
```

```
list(
  tar_eval(tar_render(name, file), values = values)
)
```

---

tar_eval_raw   *Evaluate multiple expressions created with symbol substitution (raw version).*

Description

Loop over a grid of values, create an expression object from each one, and then evaluate that expression. Helps with general metaprogramming. Unlike [tar_sub()](#), which quotes the expr argument, tar_sub_raw() assumes expr is an expression object.

Usage

```
tar_eval_raw(expr, values, envir = parent.frame())
```

Arguments

expr   Expression object with the starting expression. Values are iteratively substituted in place of symbols in expr to create each new expression, and then each expression is evaluated.

values   List of values to substitute into expr to create the expressions. All elements of values must have the same length.

envir   Environment in which to evaluate the new expressions.

Value

A list of return values from evaluating the expression objects. Often, these values are target objects. See the "Target objects" section for background on target objects specifically.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Metaprogramming utilities: [tar_eval()](#), [tar_sub()](#), [tar_sub_raw()](#)

Examples

```
# tar_map() is incompatible with tar_render() because the latter
# operates on preexisting tar_target() objects. By contrast,
# tar_eval_raw() and tar_sub_raw() iterate over code farther upstream.
values <- list(
  name = lapply(c("name1", "name2"), as.symbol),
  file = c("file1.Rmd", "file2.Rmd")
)
tar_sub_raw(quote(list(name, file)), values = values)
tar_sub_raw(quote(tar_render(name, file)), values = values)
path <- tempfile()
file.create(path)
str(tar_eval_raw(quote(tar_render(name, path)), values = values))
# So in your _targets.R file, you can define a pipeline like as below.
# Just make sure to set a unique name for each target
# (which tar_map() does automatically).
values <- list(
  name = lapply(c("name1", "name2"), as.symbol),
  file = c(path, path)
)
list(
  tar_eval_raw(quote(tar_render(name, file)), values = values)
)
```

---

tar_files   *Dynamic branching over output or input files.*

Description

Dynamic branching over output or input files.

Usage

```
tar_files(
  name,
  command,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = c("file", "file_fast", "url", "aws_file"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
```

```
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run [`tar_seed_set()`](#) on the result to locally recreate the target's initial RNG state.
command	R code to run the target.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Character of length 1. Must be `"file"`, `"url"`, or `"aws_file"`. See the `format` argument of `targets::tar_target()` for details.
repository	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of [`tar_resources_aws()`](#), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.
iteration	Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of [tar_target()](#) must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [tar_group()](#) function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

                 Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date. Only applies to the downstream target. The upstream target always runs.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_files() creates a pair of targets, one upstream and one downstream. The upstream target does some work and returns some file paths, and the downstream target is a pattern that applies format = "file", format = "file_fast", or format = "url". (URLs are input-only, they must already exist beforehand.) This is the correct way to dynamically iterate over file/url targets. It makes sure any downstream patterns only rerun some of their branches if the files/urls change. For more information, visit https://github.com/ropensci/targets/issues/136 and https://github.com/ropensci/drake/issues/1302.

Value

A list of two targets, one upstream and one downstream. The upstream one does some work and returns some file paths, and the downstream target is a pattern that applies format = "file", format

= "file_fast", or format = "url". See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Dynamic branching over files: tar_files_input(), tar_files_input_raw(), tar_files_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Do not use temp files in real projects
  # or else your targets will always rerun.
  paths <- unlist(replicate(2, tempfile()))
  file.create(paths)
  list(
    tarchetypes::tar_files(x, paths)
  )
})
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_files_input   *Dynamic branching over input files or URLs*

Description

Dynamic branching over input files or URLs.

Usage

```
tar_files_input(
  name,
  files,
```

```
    batches = length(files),
    format = c("file", "file_fast", "url", "aws_file"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.

files
: Nonempty character vector of known existing input files to track for changes.

batches
: Positive integer of length 1, number of batches to partition the files. The default is one file per batch (maximum number of batches) which is simplest to handle but could cause a lot of overhead and consume a lot of computing resources. Consider reducing the number of batches below the number of files for heavy workloads.

format
: Character, either "file", "file_fast", or "url". See the format argument of targets::tar_target() for details.

repository
: Character of length 1, remote repository for target storage. Choices:

  - "local": file system of the local machine.
  - "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
  - "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

  Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and

	tracked for changes where it exists in the cloud. The local file is deleted after the target runs.
iteration	Character, iteration method. Must be a method supported by the iteration argument of targets::tar_target(). The iteration method for the upstream target is always "list" in order to support batching.
error	Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory	Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection	
	Logical, whether to run base::gc() just before the target runs.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date. Only applies to the downstream target. The upstream target always runs.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_files_input() is like tar_files() but more convenient when the files in question already exist and are known in advance. Whereas tar_files() always appears outdated (e.g. with tar_outdated())

because it always needs to check which files it needs to branch over, `tar_files_input()` will appear up to date if the files have not changed since last `tar_make()`. In addition, `tar_files_input()` automatically groups input files into batches to reduce overhead and increase the efficiency of parallel processing.

`tar_files_input()` creates a pair of targets, one upstream and one downstream. The upstream target does some work and returns some file paths, and the downstream target is a pattern that applies `format = "file"`, `format = "file_fast"`, or `format = "url"`. This is the correct way to dynamically iterate over file/url targets. It makes sure any downstream patterns only rerun some of their branches if the files/urls change. For more information, visit https://github.com/ropensci/targets/issues/136 and https://github.com/ropensci/drake/issues/1302.

Value

A list of two targets, one upstream and one downstream. The upstream one does some work and returns some file paths, and the downstream target is a pattern that applies `format = "file"` or `format = "url"`. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Dynamic branching over files: [tar_files()](), [tar_files_input_raw()](), [tar_files_raw()]()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Do not use temp files in real projects
  # or else your targets will always rerun.
  paths <- unlist(replicate(4, tempfile()))
  file.create(paths)
  list(
    tarchetypes::tar_files_input(
      x,
      paths,
      batches = 2
    )
  )
})
})
```

```
targets::tar_make()
targets::tar_read(x)
targets::tar_read(x, branches = 1)
})
}
```

---

tar_files_input_raw   *Dynamic branching over input files or URLs (raw version).*

Description

Dynamic branching over input files or URLs.

Usage

```
tar_files_input_raw(
  name,
  files,
  batches = length(files),
  format = c("file", "file_fast", "url", "aws_file"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                    R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                    bolically to induce a dependency relationship: e.g. tar_target(downstream_target,
                    f(upstream_target)) is a target named downstream_target which depends
                    on a target upstream_target and a function f(). In addition, a target's name
                    determines its random number generator seed. In this way, each target runs with
                    a reproducible seed so someone else running the same pipeline should get the
                    same results, and no two targets in the same pipeline share the same seed. (Even
                    dynamic branches have different names and thus different seeds.) You can re-
                    cover the seed of a completed target with tar_meta(your_target, seed) and
                    run tar_seed_set() on the result to locally recreate the target's initial RNG
                    state.

files   Nonempty character vector of known existing input files to track for changes.

batches
: Positive integer of length 1, number of batches to partition the files. The default is one file per batch (maximum number of batches) which is simplest to handle but could cause a lot of overhead and consume a lot of computing resources. Consider reducing the number of batches below the number of files for heavy workloads.

format
: Character, either "file", "file_fast", or "url". See the format argument of targets::tar_target() for details.

repository
: Character of length 1, remote repository for target storage. Choices:

    - "local": file system of the local machine.
    - "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](#), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
    - "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

    Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

iteration
: Character, iteration method. Must be a method supported by the iteration argument of targets::tar_target(). The iteration method for the upstream target is always "list" in order to support batching.

error
: Character of length 1, what to do if the target stops and throws an error. Options:

    - "stop": the whole pipeline stops and throws an error.
    - "continue": the whole pipeline keeps going.
    - "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
    - "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory
: Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
: Logical, whether to run base::gc() just before the target runs.

priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).
resources	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
cue	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date. Only applies to the downstream target. The upstream target always runs.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

`tar_files_input_raw()` is similar to `tar_files_input()` except the `name` argument must be a character string.

`tar_files_input_raw()` creates a pair of targets, one upstream and one downstream. The upstream target does some work and returns some file paths, and the downstream target is a pattern that applies `format = "file"` or `format = "url"`. This is the correct way to dynamically iterate over file/url targets. It makes sure any downstream patterns only rerun some of their branches if the files/urls change. For more information, visit https://github.com/ropensci/targets/issues/136 and https://github.com/ropensci/drake/issues/1302.

Value

A list of two targets, one upstream and one downstream. The upstream one does some work and returns some file paths, and the downstream target is a pattern that applies `format = "file"` or `format = "url"`. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Dynamic branching over files: `tar_files()`, `tar_files_input()`, `tar_files_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Do not use temp files in real projects
  # or else your targets will always rerun.
  paths <- unlist(replicate(4, tempfile()))
  file.create(paths)
  list(
    tarchetypes::tar_files_input_raw(
      "x",
      paths,
      batches = 2
    )
  )
})
targets::tar_make()
targets::tar_read(x)
targets::tar_read(x, branches = 1)
})
}
```

---

tar_files_raw   *Dynamic branching over output or input files (raw version).*

Description

Dynamic branching over output or input files.

Usage

```
tar_files_raw(
  name,
  command,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = c("file", "file_fast", "url", "aws_file"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
```

```
    description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                        R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                        bolically to induce a dependency relationship: e.g. `tar_target(downstream_target,`
                        `f(upstream_target))` is a target named `downstream_target` which depends
                        on a target `upstream_target` and a function `f()`. In addition, a target's name
                        determines its random number generator seed. In this way, each target runs with
                        a reproducible seed so someone else running the same pipeline should get the
                        same results, and no two targets in the same pipeline share the same seed. (Even
                        dynamic branches have different names and thus different seeds.) You can re-
                        cover the seed of a completed target with `tar_meta(your_target, seed)` and
                        run `tar_seed_set()` on the result to locally recreate the target's initial RNG
                        state.

command   R code to run the target.

packages   Character vector of packages to load right before the target runs or the output
                        data is reloaded for downstream targets. Use `tar_option_set()` to set pack-
                        ages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading `packages`.

format   Character of length 1. Must be `"file"`, `"url"`, or `"aws_file"`. See the `format`
                        argument of `targets::tar_target()` for details.

repository   Character of length 1, remote repository for target storage. Choices:

                        • `"local"`: file system of the local machine.
                        • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a
                          non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`,
                          but versioning capabilities may be lost in doing so. See the cloud stor-
                          age section of https://books.ropensci.org/targets/data.html for
                          details for instructions.
                        • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage sec-
                          tion of https://books.ropensci.org/targets/data.html for details for
                          instructions.

                        Note: if `repository` is not `"local"` and `format` is `"file"` then the target
                        should create a single output file. That output file is uploaded to the cloud and
                        tracked for changes where it exists in the cloud. The local file is deleted after
                        the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

                        • `"vector"`: branching happens with `vctrs::vec_slice()` and aggregation
                          happens with `vctrs::vec_c()`.
                        • `"list"`, branching happens with `[[]]` and aggregation happens with `list()`.
                        • `"group"`: `dplyr::group_by()`-like functionality to branch over subsets of
                          a non-dynamic data frame. For `iteration = "group"`, the target must not
                          by dynamic (the `pattern` argument of `tar_target()` must be left NULL).

	The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the tar_group() function to see how you can create the special tar_group column with dplyr::group_by().
error	Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory	Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.
garbage_collection	
	Logical, whether to run base::gc() just before the target runs.
deployment	Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it,

then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date. Only applies to the downstream target. The upstream target always runs.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the `names` argument of functions like [tar_make()](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

`tar_files_raw()` is similar to [tar_files()](#) except the `name` argument must be a character string and `command` must be a language object.

`tar_files_raw()` creates a pair of targets, one upstream and one downstream. The upstream target does some work and returns some file paths, and the downstream target is a pattern that applies `format = "file"`, `format = "file_fast"`, or `format = "url"`. (URLs are input-only, they must already exist beforehand.) This is the correct way to dynamically iterate over file/url targets. It makes sure any downstream patterns only rerun some of their branches if the files/urls change. For more information, visit https://github.com/ropensci/targets/issues/136 and https://github.com/ropensci/drake/issues/1302.

Value

A list of two targets, one upstream and one downstream. The upstream one does some work and returns some file paths, and the downstream target is a pattern that applies `format = "file"`, `format = "file_fast"`, or `format = "url"`. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Dynamic branching over files: `tar_files()`, `tar_files_input()`, `tar_files_input_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Do not use temp files in real projects
  # or else your targets will always rerun.
  paths <- unlist(replicate(2, tempfile()))
  file.create(paths)
  command <- as.call(list(`c`, paths))
  list(
    tarchetypes::tar_files_raw("x", command)
  )
})
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_file_read | *Track a file and read the contents.*

Description

Create a pair of targets: one to track a file with `format = "file"`, and another to read the file.

Usage

```
tar_file_read(
  name,
  command,
  read,
  tidy_eval = targets::tar_option_get("tidy_eval"),
```

```
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = targets::tar_option_get("format"),
    format_file = c("file", "file_fast"),
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.
command	R code that runs in the `format = "file"` target and returns the file to be tracked.
read	R code to read the file. Must include `!!.x` where the file path goes: for example, `read = readr::read_csv(file = !!.x, col_types = readr::cols())`.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
format_file	Storage format of the file target, either `"file"` or `"file_fast"`.

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                  Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).

resources   Object returned by tar_resources() with optional settings for high-performance
                  computing functionality, alternative data storage formats, and other optional ca-
                  pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                  Must be one of the following values:

                  • "main": the target's return value is sent back to the host machine and
                    saved/uploaded locally.
                  • "worker": the worker saves/uploads the value.
                  • "none": almost never recommended. It is only for niche situations, e.g. the
                    data needs to be loaded explicitly from another language. If you do use it,
                    then the return value of the target is totally ignored when the target ends,
                    but each downstream target still attempts to load the data file (except when
                    retrieval = "none").
                    If you select storage = "none", then the return value of the target's com-
                    mand is ignored, and the data is not saved automatically. As with dynamic
                    files (format = "file") it is the responsibility of the user to write to the
                    data store from inside the target.
                    The distinguishing feature of storage = "none" (as opposed to format =
                    "file") is that in the general case, downstream targets will automatically
                    try to load the data from the data store as a dependency. As a corollary,
                    storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                  Must be one of the following values:

                  • "main": the target's dependencies are loaded on the host machine and sent
                    to the worker before the target runs.
                  • "worker": the worker loads the targets dependencies.
                  • "none": the dependencies are not loaded at all. This choice is almost never
                    recommended. It is only for niche situations, e.g. the data needs to be
                    loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether
                  the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                  target. Descriptions appear as target labels in functions like tar_manifest()
                  and tar_visnetwork(), and they let you select subsets of targets for the names
                  argument of functions like tar_make(). For example, tar_manifest(names =
                  tar_described_as(starts_with("survival model"))) lists all the targets
                  whose descriptions start with the character string "survival model".

Value

A list of two new target objects to track a file and read the contents. See the "Target objects" section
for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described

at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  tar_file_read(data, get_path(), read_csv(file = !!.x, col_types = cols()))
})
targets::tar_manifest()
})
}
```

---

tar_force   *Target with a custom condition to force execution.*

Description

Create a target that always runs if a user-defined condition rule is met.

Usage

```
tar_force(
  name,
  command,
  force,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                bolically to induce a dependency relationship: e.g. tar_target(downstream_target,
                f(upstream_target)) is a target named downstream_target which depends
                on a target upstream_target and a function f(). In addition, a target's name
                determines its random number generator seed. In this way, each target runs with
                a reproducible seed so someone else running the same pipeline should get the
                same results, and no two targets in the same pipeline share the same seed. (Even
                dynamic branches have different names and thus different seeds.) You can re-
                cover the seed of a completed target with tar_meta(your_target, seed) and
                run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG
                state.

command   R code to run the target.

force   R code for the condition that forces a build. If it evaluates to TRUE, then your
                work will run during tar_make().

tidy_eval   Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as
                the target is defined (before tar_make()). Applies to arguments command and
                force.

packages   Character vector of packages to load right before the target runs or the output
                data is reloaded for downstream targets. Use tar_option_set() to set pack-
                ages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Optional storage format for the target's return value. With the exception of
                format = "file", each target gets a file in _targets/objects, and each format
                is a different way to save and load this file. See the "Storage formats" section
                for a detailed list of possible data storage formats.

repository   Character of length 1, remote repository for target storage. Choices:

                • "local": file system of the local machine.
                • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a
                  non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](#),
                  but versioning capabilities may be lost in doing so. See the cloud stor-
                  age section of https://books.ropensci.org/targets/data.html for
                  details for instructions.
                • "gcp": Google Cloud Platform storage bucket. See the cloud storage sec-
                  tion of https://books.ropensci.org/targets/data.html for details for
                  instructions.

                Note: if repository is not "local" and format is "file" then the target
                should create a single output file. That output file is uploaded to the cloud and
                tracked for changes where it exists in the cloud. The local file is deleted after
                the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

                • "vector": branching happens with vctrs::vec_slice() and aggregation
                  happens with vctrs::vec_c().

- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of tar_target() must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the tar_group() function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

     Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.

- "worker": the worker saves/uploads the value.

- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.

- "worker": the worker loads the targets dependencies.

- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date. Only applies to the downstream target. The upstream target always runs.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_force() creates a target that always runs when a custom condition is met. The implementation builds on top of `tar_change()`. Thus, a pair of targets is created: an upstream auxiliary target to indicate the custom condition and a downstream target that responds to it and does your work.

tar_force() does not actually use `tar_cue_force()`, and the mechanism is totally different. Because the upstream target always runs, tar_outdated() and tar_visnetwork() will always show both targets as outdated. However, tar_make() will still skip the downstream one if the upstream custom condition is not met.

Value

A list of 2 targets objects: one to indicate whether the custom condition is met, and another to respond to it and do your actual work. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other targets with custom invalidation rules: tar_change(), tar_download(), tar_skip()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_force(x, tempfile(), force = 1 > 0)
  )
})
targets::tar_make()
targets::tar_make()
})
}
```

---

tar_formats   *Target formats*

Description

Target archetypes for specialized storage formats.

Usage

```
tar_url(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
```

```
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)

tar_file(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_file_fast(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
```

```
      storage = targets::tar_option_get("storage"),
      retrieval = targets::tar_option_get("retrieval"),
      cue = targets::tar_option_get("cue"),
      description = targets::tar_option_get("description")
    )

    tar_rds(
      name,
      command,
      pattern = NULL,
      tidy_eval = targets::tar_option_get("tidy_eval"),
      packages = targets::tar_option_get("packages"),
      library = targets::tar_option_get("library"),
      repository = targets::tar_option_get("repository"),
      iteration = targets::tar_option_get("iteration"),
      error = targets::tar_option_get("error"),
      memory = targets::tar_option_get("memory"),
      garbage_collection = targets::tar_option_get("garbage_collection"),
      deployment = targets::tar_option_get("deployment"),
      priority = targets::tar_option_get("priority"),
      resources = targets::tar_option_get("resources"),
      storage = targets::tar_option_get("storage"),
      retrieval = targets::tar_option_get("retrieval"),
      cue = targets::tar_option_get("cue"),
      description = targets::tar_option_get("description")
    )

    tar_qs(
      name,
      command,
      pattern = NULL,
      tidy_eval = targets::tar_option_get("tidy_eval"),
      packages = targets::tar_option_get("packages"),
      library = targets::tar_option_get("library"),
      repository = targets::tar_option_get("repository"),
      iteration = targets::tar_option_get("iteration"),
      error = targets::tar_option_get("error"),
      memory = targets::tar_option_get("memory"),
      garbage_collection = targets::tar_option_get("garbage_collection"),
      deployment = targets::tar_option_get("deployment"),
      priority = targets::tar_option_get("priority"),
      resources = targets::tar_option_get("resources"),
      storage = targets::tar_option_get("storage"),
      retrieval = targets::tar_option_get("retrieval"),
      cue = targets::tar_option_get("cue"),
      description = targets::tar_option_get("description")
    )
```

```
tar_keras(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_torch(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_format_feather(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
```

```
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_parquet(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_fst(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
```

```
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
  )

  tar_fst_dt(
    name,
    command,
    pattern = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
  )

  tar_fst_tbl(
    name,
    command,
    pattern = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
```

```
    description = targets::tar_option_get("description")
  )

  tar_aws_file(
    name,
    command,
    pattern = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
  )

  tar_aws_rds(
    name,
    command,
    pattern = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
  )

  tar_aws_qs(
    name,
    command,
```

```
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_aws_keras(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_aws_torch(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
```

```
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)

tar_format_aws_feather(
    name,
    command,
    pattern = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)

tar_aws_parquet(
    name,
    command,
    pattern = NULL,
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    repository = targets::tar_option_get("repository"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
```

```
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)

tar_aws_fst(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)

tar_aws_fst_dt(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

```
tar_aws_fst_tbl(
  name,
  command,
  pattern = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
command	R code to run the target.
pattern	Language to define branching for a target. For example, in a pipeline with numeric vector targets x and y, tar_target(z, x + y, pattern = map(x, y)) implicitly defines branches of z that each compute x[1] + y[1], x[2] + y[2], and so on. See the user manual for details.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of tar_target() must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the tar_group() function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

        Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

        If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

        The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

These functions are shorthand for targets with specialized storage formats. For example, `tar_qs(name, fun())` is equivalent to `tar_target(name, fun(), format = "qs")`. For details on specialized storage formats, open the help file of the `targets::tar_target()` function and read about the `format` argument.

Value

A `tar_target()` object with the eponymous storage format. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script(
  list(
    tarchetypes::tar_rds(x, 1)
  )
)
targets::tar_make()
})
}
```

---

tar_group_by   *Group a data frame target by one or more variables.*

Description

Create a target that outputs a grouped data frame with `dplyr::group_by()` and `targets::tar_group()`. Downstream dynamic branching targets will iterate over the groups of rows.

Usage

```
tar_group_by(
  name,
  command,
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.
command	R code to run the target.
...	Symbols, variables in the output data frame to group by.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format

is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).

resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

    If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

    The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval	Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Value

A target object to generate a grouped data frame to allows downstream dynamic targets to branch over the groups of rows. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described

at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Grouped data frame targets: tar_group_count(), tar_group_select(), tar_group_size()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  produce_data <- function() {
    expand.grid(var1 = c("a", "b"), var2 = c("c", "d"), rep = c(1, 2, 3))
  }
  list(
    tarchetypes::tar_group_by(data, produce_data(), var1, var2),
    tar_target(group, data, pattern = map(data))
  )
})
targets::tar_make()
# Read the first row group:
targets::tar_read(group, branches = 1)
# Read the second row group:
targets::tar_read(group, branches = 2)
})
}
```

---

tar_group_count   *Group the rows of a data frame into a given number groups*

Description

Create a target that outputs a grouped data frame for downstream dynamic branching. Set the maximum number of groups using count. The number of rows per group varies but is approximately uniform.

Usage

```
tar_group_count(
  name,
  command,
  count,
```

```
    tidy_eval = targets::tar_option_get("tidy_eval"),
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = targets::tar_option_get("format"),
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run [`tar_seed_set()`](#) on the result to locally recreate the target's initial RNG state.

command
: R code to run the target.

count
: Positive integer, maximum number of row groups

tidy_eval
: Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.

packages
: Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.

library
: Character vector of library paths to try when loading `packages`.

format
: Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.

repository
: Character of length 1, remote repository for target storage. Choices:

  - `"local"`: file system of the local machine.

- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                         Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage | Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"worker"`: the worker saves/uploads the value.
- `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

  If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval | Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue | An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description | Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Value

A target object to generate a grouped data frame to allows downstream dynamic targets to branch over the groups of rows. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

See Also

Other Grouped data frame targets: tar_group_by(), tar_group_select(), tar_group_size()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  produce_data <- function() {
    expand.grid(var1 = c("a", "b"), var2 = c("c", "d"), rep = c(1, 2, 3))
  }
  list(
    tarchetypes::tar_group_count(data, produce_data(), count = 2),
    tar_target(group, data, pattern = map(data))
  )
})
targets::tar_make()
# Read the first row group:
targets::tar_read(group, branches = 1)
# Read the second row group:
targets::tar_read(group, branches = 2)
})
}
```

---

tar_group_select   *Group a data frame target with* tidyselect *semantics.*

Description

Create a target that outputs a grouped data frame with dplyr::group_by() and targets::tar_group().
Unlike tar_group_by(), tar_group_select() expects you to select grouping variables using
tidyselect semantics. Downstream dynamic branching targets will iterate over the groups of
rows.

Usage

```
tar_group_select(
  name,
  command,
  by = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
```

```
    format = targets::tar_option_get("format"),
    repository = targets::tar_option_get("repository"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    storage = targets::tar_option_get("storage"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
command	R code to run the target.
by	Tidyselect semantics to specify variables to group over. Alternatively, you can supply a character vector.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices: • "local": file system of the local machine. • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](#),

but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
    If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.
    The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Value

A target object to generate a grouped data frame to allows downstream dynamic targets to branch over the groups of rows. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specifica-

tion at <https://books.ropensci.org/targets-design/> details the structure and composition of target objects.

See Also

Other Grouped data frame targets: `tar_group_by()`, `tar_group_count()`, `tar_group_size()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  produce_data <- function() {
    expand.grid(var1 = c("a", "b"), var2 = c("c", "d"), rep = c(1, 2, 3))
  }
  list(
    tarchetypes::tar_group_select(data, produce_data(), starts_with("var")),
    tar_target(group, data, pattern = map(data))
  )
})
targets::tar_make()
# Read the first row group:
targets::tar_read(group, branches = 1)
# Read the second row group:
targets::tar_read(group, branches = 2)
})
}
```

---

tar_group_size   *Group the rows of a data frame into groups of a given size.*

Description

Create a target that outputs a grouped data frame for downstream dynamic branching. Row groups have the number of rows you supply to `size` (plus the remainder in a group of its own, if applicable.) The total number of groups varies.

Usage

```
tar_group_size(
  name,
  command,
  size,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
```

```
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.
command	R code to run the target.
size	Positive integer, maximum number of rows in each group.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                    Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.

- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).

  If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the `names` argument of functions like [tar_make()](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Value

A target object to generate a grouped data frame to allows downstream dynamic targets to branch over the groups of rows. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at <https://books.ropensci.org/targets/>. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, <https://wlandau.github.io/targetopia/contributing.html#target-factories> explains target factories (functions like this one which generate targets) and the design specification at <https://books.ropensci.org/targets-design/> details the structure and composition of target objects.

See Also

Other Grouped data frame targets: tar_group_by(), tar_group_count(), tar_group_select()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  produce_data <- function() {
    expand.grid(var1 = c("a", "b"), var2 = c("c", "d"), rep = c(1, 2, 3))
  }
  list(
    tarchetypes::tar_group_size(data, produce_data(), size = 7),
    tar_target(group, data, pattern = map(data))
  )
})
targets::tar_make()
# Read the first row group:
targets::tar_read(group, branches = 1)
# Read the second row group:
targets::tar_read(group, branches = 2)
})
}
```

---

tar_hook_before   *Hook to prepend code*

Description

Prepend R code to the commands of multiple targets.

Usage

```
tar_hook_before(targets, hook, names = NULL, set_deps = TRUE)
```

Arguments

targets	A list of target objects. The input target list can be arbitrarily nested, but it must consist entirely of target objects. In addition, the return value is a simple list where each element is a target object. All hook functions remove the nested structure of the input target list.
hook	R code to insert. When you supply code to this argument, the code is quoted (not evaluated) so there is no need to wrap it in quote(), expression(), or similar.
names	Name of targets in the target list to apply the hook. Supplied with tidyselect helpers like starts_with(), as in names = starts_with("your_prefix_"). Targets not included in names still remain in the target list, but they are not modified because the hook does not apply to them.

set_deps   Logical of length 1, whether to refresh the dependencies of each modified target
                  by scanning the newly generated target commands for dependencies. If FALSE,
                  then the target will keep the original set of dependencies it had before the hook.
                  TRUE is recommended for nearly all situations. Only use FALSE if you have a
                  specialized use case and you know what you are doing.

Value

A flattened list of target objects with the hooks applied. Even if the input target list had a nested
structure, the return value is a simple list where each element is a target object. All hook functions
remove the nested structure of the input target list.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described
at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.
ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis
pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

See Also

Other hooks: tar_hook_inner(), tar_hook_outer()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  targets <- list(
    # Nested target lists work with hooks.
    list(
      targets::tar_target(x1, task1()),
      targets::tar_target(x2, task2(x1))
    ),
    targets::tar_target(x3, task3(x2)),
    targets::tar_target(y1, task4(x3))
  )
  tarchetypes::tar_hook_before(
    targets = targets,
    hook = print("Running hook."),
    names = starts_with("x")
  )
})
targets::tar_manifest(fields = command)
})
}
```

tar_hook_inner   *Hook to wrap dependencies*

Description

In the command of each target, wrap each mention of each dependency target in an arbitrary R expression.

Usage

```
tar_hook_inner(targets, hook, names = NULL, names_wrap = NULL, set_deps = TRUE)
```

Arguments

targets	A list of target objects. The input target list can be arbitrarily nested, but it must consist entirely of target objects. In addition, the return value is a simple list where each element is a target object. All hook functions remove the nested structure of the input target list.
hook	R code to wrap each target's command. The hook must contain the special placeholder symbol .x so tar_hook_inner() knows where to insert the code to wrap mentions of dependencies. The hook code is quoted (not evaluated) so there is no need to wrap it in quote(), expression(), or similar.
names	Name of targets in the target list to apply the hook. Supplied with tidyselect helpers like [starts_with()](#), as in names = starts_with("your_prefix_"). Targets not included in names still remain in the target list, but they are not modified because the hook does not apply to them.
names_wrap	Names of targets to wrap with the hook where they appear as dependencies in the commands of other targets. Use tidyselect helpers like [starts_with()](#), as in names_wrap = starts_with("your_prefix_").
set_deps	Logical of length 1, whether to refresh the dependencies of each modified target by scanning the newly generated target commands for dependencies. If FALSE, then the target will keep the original set of dependencies it had before the hook. TRUE is recommended for nearly all situations. Only use FALSE if you have a specialized use case and you know what you are doing.

Details

The expression you supply to hook must contain the special placeholder symbol .x so tar_hook_inner() knows where to insert the original command of the target.

Value

A flattened list of target objects with the hooks applied. Even if the input target list had a nested structure, the return value is a simple list where each element is a target object. All hook functions remove the nested structure of the input target list.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other hooks: `tar_hook_before()`, `tar_hook_outer()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  targets <- list(
    # Nested target lists work with hooks.
    list(
      targets::tar_target(x1, task1()),
      targets::tar_target(x2, task2(x1))
    ),
    targets::tar_target(x3, task3(x2, x1)),
    targets::tar_target(y1, task4(x3))
  )
  tarchetypes::tar_hook_inner(
    targets = targets,
    hook = fun(.x),
    names = starts_with("x")
  )
})
targets::tar_manifest(fields = command)
})
}
```

---

tar_hook_outer   *Hook to wrap commands*

Description

Wrap the command of each target in an arbitrary R expression.

Usage

```
tar_hook_outer(targets, hook, names = NULL, set_deps = TRUE)
```

Arguments

targets	A list of target objects. The input target list can be arbitrarily nested, but it must consist entirely of target objects. In addition, the return value is a simple list where each element is a target object. All hook functions remove the nested structure of the input target list.
hook	R code to wrap each target's command. The hook must contain the special placeholder symbol .x so tar_hook_outer() knows where to insert the original command of the target. The hook code is quoted (not evaluated) so there is no need to wrap it in quote(), expression(), or similar.
names	Name of targets in the target list to apply the hook. Supplied with tidyselect helpers like starts_with(), as in names = starts_with("your_prefix_"). Targets not included in names still remain in the target list, but they are not modified because the hook does not apply to them.
set_deps	Logical of length 1, whether to refresh the dependencies of each modified target by scanning the newly generated target commands for dependencies. If FALSE, then the target will keep the original set of dependencies it had before the hook. TRUE is recommended for nearly all situations. Only use FALSE if you have a specialized use case and you know what you are doing.

Details

The expression you supply to hook must contain the special placeholder symbol .x so tar_hook_outer() knows where to insert the original command of the target.

Value

A flattened list of target objects with the hooks applied. Even if the input target list had a nested structure, the return value is a simple list where each element is a target object. All hook functions remove the nested structure of the input target list.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other hooks: tar_hook_before(), tar_hook_inner()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  targets <- list(
    # Nested target lists work with hooks.
    list(
      targets::tar_target(x1, task1()),
      targets::tar_target(x2, task2(x1))
    ),
    targets::tar_target(x3, task3(x2)),
    targets::tar_target(y1, task4(x3))
  )
  tarchetypes::tar_hook_outer(
    targets = targets,
    hook = postprocess(.x, arg = "value"),
    names = starts_with("x")
  )
})
targets::tar_manifest(fields = command)
})
}
```

---

tar_knit   *Target with a* knitr *document.*

Description

Shorthand to include knitr document in a targets pipeline.

Usage

```
tar_knit(
  name,
  path,
  output_file = NULL,
  working_directory = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = "main",
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
```

```
    description = targets::tar_option_get("description"),
    quiet = TRUE,
    ...
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.

path
: Character string, file path to the knitr source file. Must have length 1.

output_file
: Character string, file path to the rendered output file.

working_directory
: Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

tidy_eval
: Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.

packages
: Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library
: Character vector of library paths to try when loading packages.

error
: Character of length 1, what to do if the target stops and throws an error. Options:

  - "stop": the whole pipeline stops and throws an error.
  - "continue": the whole pipeline keeps going.
  - "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
  - "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                        memory until the end of the pipeline (unless storage is "worker", in which case
                        targets unloads the value from memory right after storing it in order to avoid
                        sending copious data over a network). If "transient", the target gets unloaded
                        after every new target completes. Either way, the target gets automatically
                        loaded into memory whenever another target needs the value. For cloud-based
                        dynamic files (e.g. format = "file" with repository = "aws"), this memory
                        strategy applies to the temporary local copy of the file: "persistent" means
                        it remains until the end of the pipeline and is then deleted, and "transient"
                        means it gets deleted as soon as possible. The former conserves bandwidth, and
                        the latter conserves local storage.

garbage_collection
                        Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                        central controlling R process. Otherwise, if deployment is "worker" and you
                        set up the pipeline with distributed/parallel computing, then the target runs on a
                        parallel worker. For more on distributed/parallel computing in targets, please
                        visit <https://books.ropensci.org/targets/crew.html>.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                        when multiple competing targets are ready simultaneously. Targets with priori-
                        ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                        computing functionality, alternative data storage formats, and other optional ca-
                        pabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                        Must be one of the following values:

                        • "main": the target's dependencies are loaded on the host machine and sent
                          to the worker before the target runs.
                        • "worker": the worker loads the targets dependencies.
                        • "none": the dependencies are not loaded at all. This choice is almost never
                          recommended. It is only for niche situations, e.g. the data needs to be
                          loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether
                        the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                        target. Descriptions appear as target labels in functions like tar_manifest()
                        and tar_visnetwork(), and they let you select subsets of targets for the names
                        argument of functions like tar_make(). For example, tar_manifest(names =
                        tar_described_as(starts_with("survival model"))) lists all the targets
                        whose descriptions start with the character string "survival model".

quiet   Boolean; suppress the progress bar and messages?

...   Named arguments to knitr::knit(). These arguments are evaluated when the
                        target actually runs in tar_make(), not when the target is defined.

Details

`tar_knit()` is an alternative to `tar_target()` for `knitr` reports that depend on other targets. The `knitr` source should mention dependency targets with `tar_load()` and `tar_read()` in the active code chunks (which also allows you to knit the report outside the pipeline if the `_targets/` data store already exists). (Do not use `tar_load_raw()` or `tar_read_raw()` for this.) Then, `tar_knit()` defines a special kind of target. It 1. Finds all the `tar_load()`/`tar_read()` dependencies in the report and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use `tar_load_raw()` or `tar_read_raw()` for this.) 2. Sets `format = "file"` (see `tar_target()`) so `targets` watches the files at the returned paths and reruns the report if those files change. 3. Configures the target's command to return both the output report files and the input source file. All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as `deployment = "main"` in the target and `quiet = TRUE` in `knitr::knit()`.

Value

A `tar_target()` object with `format = "file"`. When this target runs, it returns a character vector of file paths. The first file paths are the output files (returned by `knitr::knit()`) and the `knitr` source file is last. But unlike `knitr::knit()`, all returned paths are *relative* paths to ensure portability (so that the project can be moved from one file system to another without invalidating the target). See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Literate programming targets: `tar_knit_raw()`, `tar_quarto()`, `tar_quarto_raw()`, `tar_quarto_rep()`, `tar_quarto_rep_raw()`, `tar_render()`, `tar_render_raw()`, `tar_render_rep()`, `tar_render_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Ordinarily, you should create the report outside
  # tar_script() and avoid temporary files.
  lines <- c(
    "---",
```

```
    "title: report",
    "output_format: html_document",
    "---",
    "",
    "```{r}",
    "targets::tar_read(data)",
    "```"
  )
  path <- tempfile()
  writeLines(lines, path)
  list(
    targets::tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tarchetypes::tar_knit(report, path)
  )
})
targets::tar_make()
})
}
```

---

tar_knitr_deps   *List literate programming dependencies.*

Description

List the target dependencies of one or more literate programming reports (R Markdown or `knitr`).

Usage

```
tar_knitr_deps(path)
```

Arguments

path   Character vector, path to one or more R Markdown or `knitr` reports.

Value

Character vector of the names of targets that are dependencies of the `knitr` report.

See Also

Other Literate programming utilities: [tar_knitr_deps_expr()](#), [tar_quarto_files()](#)

Examples

```
lines <- c(
  "---",
  "title: report",
  "output_format: html_document",
  "---",
  "",
```

```
  "```{r}",
  "targets::tar_load(data1)",
  "targets::tar_read(data2)",
  "```"
)
report <- tempfile()
writeLines(lines, report)
tar_knitr_deps(report)
```

---

tar_knitr_deps_expr   *Expression with literate programming dependencies.*

Description

Construct an expression whose global variable dependencies are the target dependencies of one or more literate programming reports (R Markdown or knitr). This helps third-party developers create their own third-party target factories for literate programming targets (similar to `tar_knit()` and `tar_render()`).

Usage

```
tar_knitr_deps_expr(path)
```

Arguments

path   Character vector, path to one or more R Markdown or knitr reports.

Value

Expression object to name the dependency targets of the knitr report, which will be detected in the static code analysis of targets.

See Also

Other Literate programming utilities: `tar_knitr_deps()`, `tar_quarto_files()`

Examples

```
lines <- c(
  "---",
  "title: report",
  "output_format: html_document",
  "---",
  "",
  "```{r}",
  "targets::tar_load(data1)",
  "targets::tar_read(data2)",
  "```"
)
```

```
report <- tempfile()
writeLines(lines, report)
tar_knitr_deps_expr(report)
```

---

tar_knit_raw *Target with a knitr document (raw version).*

Description

Shorthand to include a knitr document in a `targets` pipeline (raw version)

Usage

```
tar_knit_raw(
  name,
  path,
  output_file = NULL,
  working_directory = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = "main",
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description"),
  quiet = TRUE,
  knit_arguments = quote(list())
)
```

Arguments

name   Character of length 1, name of the target.

path   Character string, file path to the `knitr` source file. Must have length 1.

output_file   Character string, file path to the rendered output file.

working_directory

Optional character string, path to the working directory to temporarily set when running the report. The default is `NULL`, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than `NULL`, you must manually set the value of the `store` argument relative to the working directory in all calls to `tar_read()` and `tar_load()` in the report. Otherwise, these functions will not know where to find the data.

packages   Character vector of packages to load right before the target runs or the output
                    data is reloaded for downstream targets. Use tar_option_set() to set pack-
                    ages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

error   Character of length 1, what to do if the target stops and throws an error. Options:

   • "stop": the whole pipeline stops and throws an error.
   • "continue": the whole pipeline keeps going.
   • "abridge": any currently running targets keep running, but no new tar-
     gets launch after that. (Visit https://books.ropensci.org/targets/
     debugging.html to learn how to debug targets using saved workspaces.)
   • "null": The errored target continues and returns NULL. The data hash is
     deliberately wrong so the target is not up to date for the next run of the
     pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                    memory until the end of the pipeline (unless storage is "worker", in which case
                    targets unloads the value from memory right after storing it in order to avoid
                    sending copious data over a network). If "transient", the target gets unloaded
                    after every new target completes. Either way, the target gets automatically
                    loaded into memory whenever another target needs the value. For cloud-based
                    dynamic files (e.g. format = "file" with repository = "aws"), this memory
                    strategy applies to the temporary local copy of the file: "persistent" means
                    it remains until the end of the pipeline and is then deleted, and "transient"
                    means it gets deleted as soon as possible. The former conserves bandwidth, and
                    the latter conserves local storage.

garbage_collection
                    Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                    central controlling R process. Otherwise, if deployment is "worker" and you
                    set up the pipeline with distributed/parallel computing, then the target runs on a
                    parallel worker. For more on distributed/parallel computing in targets, please
                    visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                    when multiple competing targets are ready simultaneously. Targets with priori-
                    ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                    computing functionality, alternative data storage formats, and other optional ca-
                    pabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                    Must be one of the following values:

   • "main": the target's dependencies are loaded on the host machine and sent
     to the worker before the target runs.
   • "worker": the worker loads the targets dependencies.
   • "none": the dependencies are not loaded at all. This choice is almost never
     recommended. It is only for niche situations, e.g. the data needs to be
     loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether
                   the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                   target. Descriptions appear as target labels in functions like [tar_manifest()](#)
                   and [tar_visnetwork()](#), and they let you select subsets of targets for the names
                   argument of functions like [tar_make()](#). For example, tar_manifest(names =
                   tar_described_as(starts_with("survival model"))) lists all the targets
                   whose descriptions start with the character string "survival model".

quiet   Boolean; suppress the progress bar and messages?

knit_arguments   Optional language object with a list of named arguments to knitr::knit().
                   Cannot be an expression object. (Use quote(), not expression().) The reason
                   for quoting is that these arguments may depend on upstream targets whose val-
                   ues are not available at the time the target is defined, and because tar_knit_raw()
                   is the "raw" version of a function, we want to avoid all non-standard evaluation.

Details

tar_knit_raw() is just like tar_knit() except that it uses standard evaluation. The name argu-
ment is a character vector, and the knit_arguments argument is a language object.

Value

A tar_target() object with format = "file". When this target runs, it returns a character vector
of file paths. The first file paths are the output files (returned by knitr::knit()) and the knitr
source file is last. But unlike knitr::knit(), all returned paths are *relative* paths to ensure porta-
bility (so that the project can be moved from one file system to another without invalidating the
target). See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described
at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.
ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis
pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

See Also

Other Literate programming targets: [tar_knit()](#), [tar_quarto()](#), [tar_quarto_raw()](#), [tar_quarto_rep()](#),
[tar_quarto_rep_raw()](#), [tar_render()](#), [tar_render_raw()](#), [tar_render_rep()](#), [tar_render_rep_raw()](#)

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Ordinarily, you should create the report outside
  # tar_script() and avoid temporary files.
  lines <- c(
    "---",
    "title: report",
    "output_format: html_document",
    "---",
    "",
    "```{r}",
    "targets::tar_read(data)",
    "```"
  )
  path <- tempfile()
  writeLines(lines, path)
  list(
    targets::tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tarchetypes::tar_knit_raw("report", path)
  )
})
targets::tar_make()
})
}
```

---

tar_map   *Static branching.*

Description

Define multiple new targets based on existing target objects.

Usage

```
tar_map(
  values,
  ...,
  names = tidyselect::everything(),
  descriptions = tidyselect::everything(),
  unlist = FALSE
)
```

Arguments

values   Named list or data frame with values to iterate over. The names are the names
                of symbols in the commands and pattern statements, and the elements are values
                that get substituted in place of those symbols. `tar_map()` uses these elements to

create new R code, so they should be basic types, symbols, or R expressions. For objects even a little bit complicated, especially objects with attributes, it is not obvious how to convert the object into code that generates it. For complicated objects, consider using quote() when you define values, as shown at https://github.com/ropensci/tarchetypes/discussions/105.

...   One or more target objects or list of target objects. Lists can be arbitrarily nested, as in list().

names   Subset of names(values) used to generate the suffixes in the names of the new targets. The value of names should be a tidyselect expression such as a call to any_of() or starts_with().

descriptions   Names of a column in values to append to the custom description of each generated target. The value of descriptions should be a tidyselect expression such as a call to any_of() or starts_with().

unlist   Logical, whether to flatten the returned list of targets. If unlist = FALSE, the list is nested and sub-lists are named and grouped by the original input targets. If unlist = TRUE, the return value is a flat list of targets named by the new target names.

Details

tar_map() creates collections of new targets by iterating over a list of arguments and substituting symbols into commands and pattern statements.

Value

A list of new target objects. If unlist is FALSE, the list is nested and sub-lists are named and grouped by the original input targets. If unlist = TRUE, the return value is a flat list of targets named by the new target names. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other branching: tar_combine(), tar_combine_raw(), tar_map2(), tar_map2_count(), tar_map2_count_raw(), tar_map2_raw(), tar_map2_size(), tar_map2_size_raw(), tar_map_rep(), tar_map_rep_raw(), tar_rep(), tar_rep2(), tar_rep2_raw(), tar_rep_map(), tar_rep_map_raw(), tar_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_map(
      list(a = c(12, 34), b = c(45, 78)),
      targets::tar_target(x, a + b),
      targets::tar_target(y, x + a, pattern = map(x))
    )
  )
})
targets::tar_manifest()
})
}
```

---

tar_map2_count   *Dynamic-within-static branching for data frames (count batching).*

Description

Define targets for batched dynamic-within-static branching for data frames, where the user sets the (maximum) number of batches.

Usage

```
tar_map2_count(
  name,
  command1,
  command2,
  values = NULL,
  names = NULL,
  descriptions = tidyselect::everything(),
  batches = 1L,
  combine = TRUE,
  suffix1 = "1",
  suffix2 = "2",
  columns1 = tidyselect::everything(),
  columns2 = tidyselect::everything(),
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
```

```
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.

command1
: R code to create named arguments to command2. Must return a data frame with one row per call to command2.

command2
: R code to map over the data frame of arguments produced by command1. Must return a data frame.

values
: Named list or data frame with values to iterate over. The names are the names of symbols in the commands and pattern statements, and the elements are values that get substituted in place of those symbols. [tar_map()](#) uses these elements to create new R code, so they should be basic types, symbols, or R expressions. For objects even a little bit complicated, especially objects with attributes, it is not obvious how to convert the object into code that generates it. For complicated objects, consider using quote() when you define values, as shown at https://github.com/ropensci/tarchetypes/discussions/105.

names
: Language object with a tidyselect expression to select which columns of values to use to construct statically branched target names. If NULL, then short names are automatically generated.

descriptions
: Names of a column in values to append to the custom description of each generated target. The value of descriptions should be a tidyselect expression such as a call to [any_of()](#) or [starts_with()](#).

batches
: Number of batches. This is also the number of dynamic branches created during tar_make().

combine
: Logical of length 1, whether to statically combine all the results into a single target downstream.

suffix1
: Character of length 1, suffix to apply to the command1 targets to distinguish them from the command2 targets.

suffix2	Character of length 1, suffix to apply to the `command2` targets to distinguish them from the `command1` targets.
columns1	A tidyselect expression to select which columns of `values` to append to the output of all targets. Columns already in the target output are not appended.
columns2	A tidyselect expression to select which columns of `command1` output to append to `command2` output. Columns already in the target output are not appended. `columns1` takes precedence over `columns2`.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
tidy_eval	Whether to invoke tidy evaluation (e.g. the `!!` operator from `rlang`) as soon as the target is defined (before `tar_make()`). Applies to the `command` argument.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of [`tar_resources_aws()`](), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error	Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- `"null"`: The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory	Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.
garbage_collection	
	Logical, whether to run base::gc() just before the target runs.
deployment	Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.

- "worker": the worker loads the targets dependencies.

- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Static branching creates one pair of targets for each row in values. In each pair, there is an upstream non-dynamic target that runs command1 and a downstream dynamic target that runs command2. command1 produces a data frame of arguments to command2, and command2 dynamically maps over these arguments in batches.

Value

A list of new target objects. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why [tar_rep()](#) and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are [tar_rep2()](#), [tar_map_rep()](#), [tar_map2_count()](#), [tar_map2_size()](#), and [tar_render_rep()](#).

For the tar_map2_*() functions, it is possible to manually supply your own seeds through the
command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed,
or withr::local_seed()). For tar_render_rep(), custom seeds can be supplied to the params
argument and then invoked in the individual R Markdown reports. Likewise with tar_quarto_rep()
and the execute_params argument.

See Also

Other branching: tar_combine(), tar_combine_raw(), tar_map(), tar_map2(), tar_map2_count_raw(),
tar_map2_raw(), tar_map2_size(), tar_map2_size_raw(), tar_map_rep(), tar_map_rep_raw(),
tar_rep(), tar_rep2(), tar_rep2_raw(), tar_rep_map(), tar_rep_map_raw(), tar_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  tarchetypes::tar_map2_count(
    x,
    command1 = tibble::tibble(
      arg1 = arg1,
      arg2 = seq_len(6)
     ),
    command2 = tibble::tibble(
      result = paste(arg1, arg2),
      random = sample.int(1e9, size = 1),
      length_input = length(arg1)
    ),
    values = tibble::tibble(arg1 = letters[seq_len(2)]),
    batches = 3
   )
})
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_map2_count_raw   *Dynamic-within-static branching for data frames (count batching;*
                           raw version).

Description

Define targets for batched dynamic-within-static branching for data frames, where the user sets the
(maximum) number of batches. Like tar_map2_count() except name is a character string and
command1, command2, names, columns1, and columns2 are all language objects.

Usage

```
tar_map2_count_raw(
  name,
  command1,
  command2,
  values = NULL,
  names = NULL,
  descriptions = quote(tidyselect::everything()),
  batches = 1L,
  combine = TRUE,
  suffix1 = "1",
  suffix2 = "2",
  rep_workers = 1,
  columns1 = quote(tidyselect::everything()),
  columns2 = quote(tidyselect::everything()),
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.
command1	Language object to create named arguments to `command2`. Must return a data frame with one row per call to `command2`.

command2	Language object to map over the data frame of arguments produced by command1. Must return a data frame.
values	Named list or data frame with values to iterate over. The names are the names of symbols in the commands and pattern statements, and the elements are values that get substituted in place of those symbols. `tar_map()` uses these elements to create new R code, so they should be basic types, symbols, or R expressions. For objects even a little bit complicated, especially objects with attributes, it is not obvious how to convert the object into code that generates it. For complicated objects, consider using quote() when you define values, as shown at https://github.com/ropensci/tarchetypes/discussions/105.
names	Language object with a tidyselect expression to select which columns of values to use to construct statically branched target names. If NULL, then short names are automatically generated.
descriptions	Names of a column in values to append to the custom description of each generated target. The value of descriptions should be a tidyselect expression such as a call to any_of() or starts_with().
batches	Positive integer of length 1, maximum number of batches (dynamic branches within static branches) of the downstream (command2) targets. Batches are formed from row groups of the command1 target output.
combine	Logical of length 1, whether to statically combine all the results into a single target downstream.
suffix1	Character of length 1, suffix to apply to the command1 targets to distinguish them from the command2 targets.
suffix2	Character of length 1, suffix to apply to the command2 targets to distinguish them from the command1 targets.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
columns1	Language object, a tidyselect expression to select which columns of values to append to the output of all targets.
columns2	Language object, a tidyselect expression to select which columns of command1 output to append to command2 output. In case of conflicts, column1 takes precedence.
tidy_eval	Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as the target is defined (before tar_make()). Applies to the command argument.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if `repository` is not "local" and `format` is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error
: Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory
: Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless `storage` is "worker", in which case `targets` unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. `format = "file"` with `repository = "aws"`), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
: Logical, whether to run `base::gc()` just before the target runs.

deployment
: Character of length 1. If `deployment` is "main", then the target will run on the central controlling R process. Otherwise, if `deployment` is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in `targets`, please visit https://books.ropensci.org/targets/crew.html.

priority
: Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).

resources
: Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.

storage   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"worker"`: the worker saves/uploads the value.
- `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).
  If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.
  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

Static branching creates one pair of targets for each row in `values`. In each pair, there is an upstream non-dynamic target that runs `command1` and a downstream dynamic target that runs `command2`. `command1` produces a data frame of arguments to `command2`, and `command2` dynamically maps over these arguments in batches.

Value

A list of new target objects. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described
at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.
ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis
pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number gen-
erator seed derived from its target name. In batched replicate, however, each batch is a target
with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate*
its own unique seed. Each replicate-specific seed is created based on the dynamic parent target
name, tar_option_get("seed") (for `targets` version 0.13.5.9000 and above), batch index, and
rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are in-
variant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches =
100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1),
batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this
seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`.
For the tar_map2_*() functions, it is possible to manually supply your own seeds through the
command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed,
or withr::local_seed()). For `tar_render_rep()`, custom seeds can be supplied to the params
argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()`
and the execute_params argument.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`,
`tar_map2_raw()`, `tar_map2_size()`, `tar_map2_size_raw()`, `tar_map_rep()`, `tar_map_rep_raw()`,
`tar_rep()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  tarchetypes::tar_map2_count_raw(
    "x",
    command1 = quote(
      tibble::tibble(
        arg1 = arg1,
        arg2 = seq_len(6)
      )
    ),
    command2 = quote(
```

```
        tibble::tibble(
          result = paste(arg1, arg2),
          random = sample.int(1e6, size = 1),
          length_input = length(arg1)
        )
      ),
      values = tibble::tibble(arg1 = letters[seq_len(2)]),
      batches = 3
    )
  })
  targets::tar_make()
  targets::tar_read(x)
})
}
```

---

tar_map2_size   *Dynamic-within-static branching for data frames (size batching).*

Description

Define targets for batched dynamic-within-static branching for data frames, where the user sets the (maximum) size of each batch.

Usage

```
tar_map2_size(
  name,
  command1,
  command2,
  values = NULL,
  names = NULL,
  descriptions = tidyselect::everything(),
  size = Inf,
  combine = TRUE,
  suffix1 = "1",
  suffix2 = "2",
  columns1 = tidyselect::everything(),
  columns2 = tidyselect::everything(),
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
```

```
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
command1	R code to create named arguments to command2. Must return a data frame with one row per call to command2.
command2	R code to map over the data frame of arguments produced by command1. Must return a data frame.
values	Named list or data frame with values to iterate over. The names are the names of symbols in the commands and pattern statements, and the elements are values that get substituted in place of those symbols. [tar_map()](#) uses these elements to create new R code, so they should be basic types, symbols, or R expressions. For objects even a little bit complicated, especially objects with attributes, it is not obvious how to convert the object into code that generates it. For complicated objects, consider using quote() when you define values, as shown at https://github.com/ropensci/tarchetypes/discussions/105.
names	Language object with a tidyselect expression to select which columns of values to use to construct statically branched target names. If NULL, then short names are automatically generated.
descriptions	Names of a column in values to append to the custom description of each generated target. The value of descriptions should be a tidyselect expression such as a call to [any_of()](#) or [starts_with()](#).
size	Positive integer of length 1, maximum number of rows in each batch for the downstream (command2) targets. Batches are formed from row groups of the command1 target output.
combine	Logical of length 1, whether to statically combine all the results into a single target downstream.
suffix1	Character of length 1, suffix to apply to the command1 targets to distinguish them from the command2 targets.

suffix2	Character of length 1, suffix to apply to the command2 targets to distinguish them from the command1 targets.
columns1	A tidyselect expression to select which columns of values to append to the output of all targets. Columns already in the target output are not appended.
columns2	A tidyselect expression to select which columns of command1 output to append to command2 output. Columns already in the target output are not appended. columns1 takes precedence over columns2.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
tidy_eval	Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as the target is defined (before tar_make()). Applies to the command argument.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices: • "local": file system of the local machine. • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.
error	Character of length 1, what to do if the target stops and throws an error. Options: • "stop": the whole pipeline stops and throws an error. • "continue": the whole pipeline keeps going. • "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.) • "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                        memory until the end of the pipeline (unless storage is "worker", in which case
                        targets unloads the value from memory right after storing it in order to avoid
                        sending copious data over a network). If "transient", the target gets unloaded
                        after every new target completes. Either way, the target gets automatically
                        loaded into memory whenever another target needs the value. For cloud-based
                        dynamic files (e.g. format = "file" with repository = "aws"), this memory
                        strategy applies to the temporary local copy of the file: "persistent" means
                        it remains until the end of the pipeline and is then deleted, and "transient"
                        means it gets deleted as soon as possible. The former conserves bandwidth, and
                        the latter conserves local storage.

garbage_collection
                        Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                        central controlling R process. Otherwise, if deployment is "worker" and you
                        set up the pipeline with distributed/parallel computing, then the target runs on a
                        parallel worker. For more on distributed/parallel computing in targets, please
                        visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                        when multiple competing targets are ready simultaneously. Targets with priori-
                        ties closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance
                        computing functionality, alternative data storage formats, and other optional ca-
                        pabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                        Must be one of the following values:

                          • "main": the target's return value is sent back to the host machine and
                            saved/uploaded locally.
                          • "worker": the worker saves/uploads the value.
                          • "none": almost never recommended. It is only for niche situations, e.g. the
                            data needs to be loaded explicitly from another language. If you do use it,
                            then the return value of the target is totally ignored when the target ends,
                            but each downstream target still attempts to load the data file (except when
                            retrieval = "none").
                            If you select storage = "none", then the return value of the target's com-
                            mand is ignored, and the data is not saved automatically. As with dynamic
                            files (format = "file") it is the responsibility of the user to write to the
                            data store from inside the target.
                            The distinguishing feature of storage = "none" (as opposed to format =
                            "file") is that in the general case, downstream targets will automatically
                            try to load the data from the data store as a dependency. As a corollary,
                            storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future().
                        Must be one of the following values:

                          • "main": the target's dependencies are loaded on the host machine and sent
                            to the worker before the target runs.

- "worker": the worker loads the targets dependencies.

- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

Static branching creates one pair of targets for each row in values. In each pair, there is an upstream non-dynamic target that runs command1 and a downstream dynamic target that runs command2. command1 produces a data frame of arguments to command2, and command2 dynamically maps over these arguments in batches.

Value

A list of new target objects. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why tar_rep() and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are tar_rep2(), tar_map_rep(), tar_map2_count(), tar_map2_size(), and tar_render_rep().

For the `tar_map2_*()` functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (`set.seed()`, `withr::with_seed`, or `withr::local_seed()`). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the execute_params argument.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`, `tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size_raw()`, `tar_map_rep()`, `tar_map_rep_raw()`, `tar_rep()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  tarchetypes::tar_map2_size(
    x,
    command1 = tibble::tibble(
      arg1 = arg1,
      arg2 = seq_len(6)
    ),
    command2 = tibble::tibble(
      result = paste(arg1, arg2),
      random = sample.int(1e9, size = 1),
      length_input = length(arg1)
    ),
    values = tibble::tibble(arg1 = letters[seq_len(2)]),
    size = 2
  )
})
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_map2_size_raw   *Dynamic-within-static branching for data frames (size batching; raw*
                           version).

Description

Define targets for batched dynamic-within-static branching for data frames, where the user sets the (maximum) size of each batch. Like `tar_map2_size()` except name is a character string and command1, command2, names, columns1, and columns2 are all language objects.

Usage

```
tar_map2_size_raw(
  name,
  command1,
  command2,
  values = NULL,
  names = NULL,
  descriptions = quote(tidyselect::everything()),
  size = Inf,
  combine = TRUE,
  suffix1 = "1",
  suffix2 = "2",
  columns1 = quote(tidyselect::everything()),
  columns2 = quote(tidyselect::everything()),
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target,` `f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.
command1	Language object to create named arguments to `command2`. Must return a data frame with one row per call to `command2`.

command2   Language object to map over the data frame of arguments produced by command1.
                Must return a data frame.

values   Named list or data frame with values to iterate over. The names are the names
                of symbols in the commands and pattern statements, and the elements are values
                that get substituted in place of those symbols. [tar_map()](#) uses these elements to
                create new R code, so they should be basic types, symbols, or R expressions. For
                objects even a little bit complicated, especially objects with attributes, it is not
                obvious how to convert the object into code that generates it. For complicated
                objects, consider using quote() when you define values, as shown at https:
                //github.com/ropensci/tarchetypes/discussions/105.

names   Language object with a tidyselect expression to select which columns of values
                to use to construct statically branched target names. If NULL, then short names
                are automatically generated.

descriptions   Names of a column in values to append to the custom description of each gen-
                erated target. The value of descriptions should be a tidyselect expression
                such as a call to [any_of()](#) or [starts_with()](#).

size   Positive integer of length 1, maximum number of rows in each batch for the
                downstream (command2) targets. Batches are formed from row groups of the
                command1 target output.

combine   Logical of length 1, whether to statically combine all the results into a single
                target downstream.

suffix1   Character of length 1, suffix to apply to the command1 targets to distinguish them
                from the command2 targets.

suffix2   Character of length 1, suffix to apply to the command2 targets to distinguish them
                from the command1 targets.

columns1   Language object, a tidyselect expression to select which columns of values to
                append to the output of all targets.

columns2   Language object, a tidyselect expression to select which columns of command1
                output to append to command2 output. In case of conflicts, column1 takes prece-
                dence.

rep_workers   Positive integer of length 1, number of local R processes to use to run reps within
                batches in parallel. If 1, then reps are run sequentially within each batch. If
                greater than 1, then reps within batch are run in parallel using a PSOCK cluster.

tidy_eval   Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as
                the target is defined (before tar_make()). Applies to the command argument.

packages   Character vector of packages to load right before the target runs or the output
                data is reloaded for downstream targets. Use tar_option_set() to set pack-
                ages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Optional storage format for the target's return value. With the exception of
                format = "file", each target gets a file in _targets/objects, and each format
                is a different way to save and load this file. See the "Storage formats" section
                for a detailed list of possible data storage formats.

repository   Character of length 1, remote repository for target storage. Choices:

- ```"local"```: file system of the local machine.
- ```"aws"```: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the ```endpoint``` argument of ```tar_resources_aws()```, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- ```"gcp"```: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if ```repository``` is not ```"local"``` and ```format``` is ```"file"``` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- ```"stop"```: the whole pipeline stops and throws an error.
- ```"continue"```: the whole pipeline keeps going.
- ```"abridge"```: any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- ```"null"```: The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If ```"persistent"```, the target stays in memory until the end of the pipeline (unless ```storage``` is ```"worker"```, in which case ```targets``` unloads the value from memory right after storing it in order to avoid sending copious data over a network). If ```"transient"```, the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. ```format = "file"``` with ```repository = "aws"```), this memory strategy applies to the temporary local copy of the file: ```"persistent"``` means it remains until the end of the pipeline and is then deleted, and ```"transient"``` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

             Logical, whether to run ```base::gc()``` just before the target runs.

deployment   Character of length 1. If ```deployment``` is ```"main"```, then the target will run on the central controlling R process. Otherwise, if ```deployment``` is ```"worker"``` and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in ```targets```, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in ```tar_make_future()```).

resources   Object returned by ```tar_resources()``` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of ```targets```. See ```tar_resources()``` for details.

storage   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`.
                Must be one of the following values:

                - `"main"`: the target's return value is sent back to the host machine and
                  saved/uploaded locally.
                - `"worker"`: the worker saves/uploads the value.
                - `"none"`: almost never recommended. It is only for niche situations, e.g. the
                  data needs to be loaded explicitly from another language. If you do use it,
                  then the return value of the target is totally ignored when the target ends,
                  but each downstream target still attempts to load the data file (except when
                  `retrieval = "none"`).
                  If you select `storage = "none"`, then the return value of the target's com-
                  mand is ignored, and the data is not saved automatically. As with dynamic
                  files (`format = "file"`) it is the responsibility of the user to write to the
                  data store from inside the target.
                  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically
                  try to load the data from the data store as a dependency. As a corollary,
                  `storage = "none"` is completely unnecessary if `format` is `"file"`.

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`.
                Must be one of the following values:

                - `"main"`: the target's dependencies are loaded on the host machine and sent
                  to the worker before the target runs.
                - `"worker"`: the worker loads the targets dependencies.
                - `"none"`: the dependencies are not loaded at all. This choice is almost never
                  recommended. It is only for niche situations, e.g. the data needs to be
                  loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether
                the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the
                target. Descriptions appear as target labels in functions like `tar_manifest()`
                and `tar_visnetwork()`, and they let you select subsets of targets for the `names`
                argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets
                whose descriptions start with the character string `"survival model"`.

Details

Static branching creates one pair of targets for each row in `values`. In each pair, there is an upstream
non-dynamic target that runs `command1` and a downstream dynamic target that runs `command2`.
`command1` produces a data frame of arguments to `command2`, and `command2` dynamically maps over
these arguments in batches.

Value

A list of new target objects. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for `targets` version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the tar_map2_*() functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed, or withr::local_seed()). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the execute_params argument.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`, `tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size()`, `tar_map_rep()`, `tar_map_rep_raw()`, `tar_rep()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  tarchetypes::tar_map2_size_raw(
    "x",
    command1 = quote(
      tibble::tibble(
        arg1 = arg1,
        arg2 = seq_len(6)
      )
    ),
    command2 = quote(
```

```
      tibble::tibble(
        result = paste(arg1, arg2),
        random = sample.int(1e6, size = 1),
        length_input = length(arg1)
      )
    ),
    values = tibble::tibble(arg1 = letters[seq_len(2)]),
    size = 2
    )
})
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_map_rep   *Dynamic batched replication within static branches for data frames.*

Description

Define targets for batched replication within static branches for data frames.

Usage

```
tar_map_rep(
  name,
  command,
  values = NULL,
  names = NULL,
  descriptions = tidyselect::everything(),
  columns = tidyselect::everything(),
  batches = 1,
  reps = 1,
  rep_workers = 1,
  combine = TRUE,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
```

```
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
command	R code for a single replicate. Must return a data frame.
values	Named list or data frame with values to iterate over. The names are the names of symbols in the commands and pattern statements, and the elements are values that get substituted in place of those symbols. [tar_map()](#) uses these elements to create new R code, so they should be basic types, symbols, or R expressions. For objects even a little bit complicated, especially objects with attributes, it is not obvious how to convert the object into code that generates it. For complicated objects, consider using quote() when you define values, as shown at https://github.com/ropensci/tarchetypes/discussions/105.
names	Language object with a tidyselect expression to select which columns of values to use to construct statically branched target names. If NULL, then short names are automatically generated.
descriptions	Names of a column in values to append to the custom description of each generated target. The value of descriptions should be a tidyselect expression such as a call to [any_of()](#) or [starts_with()](#).
columns	A tidyselect expression to select which columns of values to append to the output. Columns already in the target output are not appended.
batches	Number of batches. This is also the number of dynamic branches created during tar_make().
reps	Number of replications in each batch. The total number of replications is batches * reps.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
combine	Logical of length 1, whether to statically combine all the results into a single target downstream.
tidy_eval	Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as the target is defined (before tar_make()). Applies to the command argument.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of tar_resources_aws(), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

                 Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

                 Logical, whether to run base::gc() just before the target runs.

deployment	Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
storage	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Value

A list of new target objects. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, `tar_option_get("seed")` (for `targets` version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, `tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...)` produces the same numerical output as `tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...)` (but with different batch names). Other target factories with this seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the `tar_map2_*()` functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (`set.seed()`, `withr::with_seed`, or `withr::local_seed()`). For `tar_render_rep()`, custom seeds can be supplied to the `params` argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the `execute_params` argument.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`, `tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size()`, `tar_map2_size_raw()`, `tar_map_rep_raw()`, `tar_rep()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Just a sketch of a Bayesian sensitivity analysis of hyperparameters:
  assess_hyperparameters <- function(sigma1, sigma2) {
    # data <- simulate_random_data() # user-defined function
    # run_model(data, sigma1, sigma2) # user-defined function
    # Mock output from the model:
```

```
      posterior_samples <- stats::rnorm(1000, 0, sigma1 + sigma2)
      tibble::tibble(
        posterior_median = median(posterior_samples),
        posterior_quantile_0.025 = quantile(posterior_samples, 0.025),
        posterior_quantile_0.975 = quantile(posterior_samples, 0.975)
      )
    }
    hyperparameters <- tibble::tibble(
      scenario = c("tight", "medium", "diffuse"),
      sigma1 = c(10, 50, 50),
      sigma2 = c(10, 5, 10)
    )
    tarchetypes::tar_map_rep(
      sensitivity_analysis,
      command = assess_hyperparameters(sigma1, sigma2),
      values = hyperparameters,
      names = tidyselect::any_of("scenario"),
      batches = 2,
      reps = 3
    )
})
targets::tar_make()
targets::tar_read(sensitivity_analysis)
})
}
```

---

tar_map_rep_raw   *Dynamic batched replication within static branches for data frames*
                                    (raw version).

Description

Define targets for batched replication within static branches for data frames (raw version).

This function is like [tar_map_rep()](#) except the name argument is a character string and the names and columns arguments are language objects.

Usage

```
tar_map_rep_raw(
  name,
  command,
  values = NULL,
  names = NULL,
  descriptions = quote(tidyselect::everything()),
  columns = quote(tidyselect::everything()),
  batches = 1,
  reps = 1,
  rep_workers = 1,
  combine = TRUE,
```

```
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.
command	Language object, R code for a single replicate. Must return a data frame.
values	Named list or data frame with values to iterate over. The names are the names of symbols in the commands and pattern statements, and the elements are values that get substituted in place of those symbols. [tar_map()](#) uses these elements to create new R code, so they should be basic types, symbols, or R expressions. For objects even a little bit complicated, especially objects with attributes, it is not obvious how to convert the object into code that generates it. For complicated objects, consider using quote() when you define values, as shown at https://github.com/ropensci/tarchetypes/discussions/105.
names	Language object with a tidyselect expression to select which columns of values to use to construct statically branched target names. If NULL, then short names are automatically generated.
descriptions	Names of a column in values to append to the custom description of each generated target. The value of descriptions should be a tidyselect expression such as a call to [any_of()](#) or [starts_with()](#).
columns	Language object with a tidyselect expression to select which columns of values to append to the output. Columns already in the target output are not appended.

batches	Number of batches. This is also the number of dynamic branches created during `tar_make()`.
reps	Number of replications in each batch. The total number of replications is `batches * reps`.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
combine	Logical of length 1, whether to statically combine all the results into a single target downstream.
tidy_eval	Whether to invoke tidy evaluation (e.g. the `!!` operator from `rlang`) as soon as the target is defined (before `tar_make()`). Applies to the `command` argument.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Character of length 1, storage format of the output. An efficient data frame format like `"feather"` is recommended, but the default is `"rds"` to avoid incurring extra package dependencies. See the help file of `targets::tar_target()` for details on storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of [`tar_resources_aws()`](tar_resources_aws()), but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error	Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- `"null"`: The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory	Character of length 1, memory strategy. If `"persistent"`, the target stays in memory until the end of the pipeline (unless `storage` is `"worker"`, in which case

targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.
  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.

- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Value

A list of new target objects. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for `targets` version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the tar_map2_*() functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed, or withr::local_seed()). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the execute_params argument.

See Also

Other branching: tar_combine(), tar_combine_raw(), tar_map(), tar_map2(), tar_map2_count(),
tar_map2_count_raw(), tar_map2_raw(), tar_map2_size(), tar_map2_size_raw(), tar_map_rep(),
tar_rep(), tar_rep2(), tar_rep2_raw(), tar_rep_map(), tar_rep_map_raw(), tar_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  # Just a sketch of a Bayesian sensitivity analysis of hyperparameters:
  assess_hyperparameters <- function(sigma1, sigma2) {
    # data <- simulate_random_data() # user-defined function
    # run_model(data, sigma1, sigma2) # user-defined function
    # Mock output from the model:
    posterior_samples <- stats::rnorm(1000, 0, sigma1 + sigma2)
    tibble::tibble(
      posterior_median = median(posterior_samples),
      posterior_quantile_0.025 = quantile(posterior_samples, 0.025),
      posterior_quantile_0.975 = quantile(posterior_samples, 0.975)
    )
  }
  hyperparameters <- tibble::tibble(
    scenario = c("tight", "medium", "diffuse"),
    sigma1 = c(10, 50, 50),
    sigma2 = c(10, 5, 10)
  )
  tarchetypes::tar_map_rep_raw(
    "sensitivity_analysis",
    command = quote(assess_hyperparameters(sigma1, sigma2)),
    values = hyperparameters,
    names = quote(tidyselect::any_of("scenario")),
    batches = 2,
    reps = 3
  )
})
targets::tar_make()
targets::tar_read(sensitivity_analysis)
})
}
```

---

tar_plan   *A* drake*-plan-like pipeline archetype*

Description

Simplify target specification in pipelines.

Usage

```
tar_plan(...)
```

Arguments

`...`	Named and unnamed targets. All named targets must follow the drake-plan-like `target = command` syntax, and all unnamed arguments must be explicit calls to create target objects, e.g. `tar_target()`, target archetypes like `tar_render()`, or similar.

Details

Allows targets with just targets and commands to be written in the pipeline as `target = command` instead of `tar_target(target, command)`. Also supports ordinary target objects if they are unnamed. `tar_plan(x = 1, y = 2, tar_target(z, 3), tar_render(r, "r.Rmd"))` is equivalent to `list(tar_target(x, 1), tar_target(y, 2), tar_target(z, 3), tar_render(r, "r.Rmd"))`. # nolint

Value

A list of `tar_target()` objects. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  library(tarchetypes)
  tar_plan(
    tarchetypes::tar_fst_tbl(data, data.frame(x = seq_len(26))),
    means = colMeans(data) # No need for tar_target() for simple cases.
  )
})
targets::tar_make()
})
}
```

---

tar_quarto   *Target with a Quarto project.*

Description

Shorthand to include a Quarto project in a `targets` pipeline.

Usage

```
tar_quarto(
  name,
  path = ".",
  working_directory = NULL,
  extra_files = character(0),
  execute = TRUE,
  execute_params = list(),
  cache = NULL,
  cache_refresh = FALSE,
  debug = FALSE,
  quiet = TRUE,
  quarto_args = NULL,
  pandoc_args = NULL,
  profile = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = NULL,
  library = NULL,
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = "main",
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                bolically to induce a dependency relationship: e.g. `tar_target(downstream_target,`
                `f(upstream_target))` is a target named `downstream_target` which depends
                on a target `upstream_target` and a function `f()`. In addition, a target's name
                determines its random number generator seed. In this way, each target runs with
                a reproducible seed so someone else running the same pipeline should get the
                same results, and no two targets in the same pipeline share the same seed. (Even

dynamic branches have different names and thus different seeds.) You can re-
cover the seed of a completed target with tar_meta(your_target, seed) and
run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG
state.

path   Character of length 1, either the single *.qmd source file to be rendered or a
                     directory containing a Quarto project. Defaults to the working directory of the
                     targets pipeline. Passed directly to the input argument of quarto::quarto_render().

working_directory
                     Optional character string, path to the working directory to temporarily set when
                     running the report. The default is NULL, which runs the report from the current
                     working directory at the time the pipeline is run. This default is recommended in
                     the vast majority of cases. To use anything other than NULL, you must manually
                     set the value of the store argument relative to the working directory in all calls
                     to tar_read() and tar_load() in the report. Otherwise, these functions will
                     not know where to find the data.

extra_files   Character vector of extra files and directories to track for changes. The target
                     will be invalidated (rerun on the next tar_make()) if the contents of these files
                     changes. No need to include anything already in the output of [tar_quarto_files()](#),
                     the list of file dependencies automatically detected through quarto::quarto_inspect().

execute   Whether to execute embedded code chunks.

execute_params   Code, cannot be NULL. execute_params evaluates to a named list of parameters
                     for parameterized Quarto documents. These parameters override the custom
                     custom elements of the params list in the YAML front-matter of the Quarto
                     source files. The list is quoted (not evaluated until the target runs) so that up-
                     stream targets can serve as parameter values.

cache   Cache execution output (uses knitr cache and jupyter-cache respectively for Rmd
                     and Jupyter input files).

cache_refresh   Force refresh of execution cache.

debug   Leave intermediate files in place after render.

quiet   Suppress warning and other messages.

quarto_args   Character vector of other quarto CLI flag pass to the command. This is mainly
                     for advanced usage, e.g it can be useful for new options added to quarto CLI and
                     not yet supported as function argument.

pandoc_args   Additional command line options to pass to pandoc.

profile   Character of length 1, Quarto profile. If NULL, the default profile will be used.
                     Requires Quarto version 1.2 or higher. See [https://quarto.org/docs/projects/
                     profiles.html](https://quarto.org/docs/projects/profiles.html) for details.

tidy_eval   Logical, whether to enable tidy evaluation when interpreting command and pattern.
                     If TRUE, you can use the "bang-bang" operator !! to programmatically insert the
                     values of global objects.

packages   Character vector of packages to load right before the target runs or the output
                     data is reloaded for downstream targets. Use tar_option_set() to set pack-
                     ages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                        Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest()

and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

`tar_quarto()` is an alternative to `tar_target()` for Quarto projects and standalone Quarto source documents that depend on upstream targets. The Quarto R source documents (*.qmd and *.Rmd files) should mention dependency targets with tar_load() and tar_read() in the active R code chunks (which also allows you to render the project outside the pipeline if the _targets/ data store already exists). (Do not use tar_load_raw() or tar_read_raw() for this.) Then, tar_quarto() defines a special kind of target. It 1. Finds all the tar_load()/tar_read() dependencies in the R source reports and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use tar_load_raw() or tar_read_raw() for this.) 2. Sets format = "file" (see tar_target()) so targets watches the files at the returned paths and reruns the report if those files change. 3. Configures the target's command to return both the output rendered files and the input dependency files (such as Quarto source documents). All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as deployment = "main" in the target and quiet = TRUE in quarto::quarto_render().

Value

A target object with format = "file". When this target runs, it returns a character vector of file paths: the rendered documents, the Quarto source files, and other input and output files. The output files are determined by the YAML front-matter of standalone Quarto documents and _quarto.yml in Quarto projects, and you can see these files with `tar_quarto_files()` (powered by quarto::quarto_inspect()). All returned paths are *relative* paths to ensure portability (so that the project can be moved from one file system to another without invalidating the target). See the "Target objects" section for background.

Quarto troubleshooting

If you encounter difficult errors, please read `https://github.com/quarto-dev/quarto-r/issues/16`. In addition, please try to reproduce the error using quarto::quarto_render("your_report.qmd", execute_dir = getwd()) without using targets at all. Isolating errors this way makes them much easier to solve.

Literate programming limitations

Literate programming files are messy and variable, so functions like `tar_render()` have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if `tar_read()` and/or `tar_load()` are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. tar_load("x") and tar_load(contains("x")) may not detect target x. * Special/optional input/output files may not be detected in all cases. * `tar_render()` and friends are for local files only. They do not integrate with the cloud storage capabilities of targets.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Literate programming targets: `tar_knit()`, `tar_knit_raw()`, `tar_quarto_raw()`, `tar_quarto_rep()`, `tar_quarto_rep_raw()`, `tar_render()`, `tar_render_raw()`, `tar_render_rep()`, `tar_render_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({  # tar_dir() runs code from a temporary directory.
# Unparameterized Quarto document:
lines <- c(
  "---",
  "title: report.qmd source file",
  "output_format: html",
  "---",
  "Assume these lines are in report.qmd.",
  "```{r}",
  "targets::tar_read(data)",
  "```"
)
writeLines(lines, "report.qmd")
# Include the report in a pipeline as follows.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tar_quarto(report, path = "report.qmd")
  )
}, ask = FALSE)
# Then, run the pipeline as usual.

# Parameterized Quarto:
lines <- c(
  "---",
  "title: 'report.qmd source file with parameters'",
  "output_format: html_document",
  "params:",
  "  your_param: \"default value\"",
  "---",
  "Assume these lines are in report.qmd.",
```

```
    "```{r}",
    "print(params$your_param)",
    "```"
  )
  writeLines(lines, "report.qmd")
  # Include the report in the pipeline as follows.
  unlink("_targets.R") # In tar_dir(), not the user's file space.
  targets::tar_script({
    library(tarchetypes)
    list(
      tar_target(data, data.frame(x = seq_len(26), y = letters)),
      tar_quarto(
        report,
        path = "report.qmd",
        execute_params = list(your_param = data)
      )
    )
  }, ask = FALSE)
})
# Then, run the pipeline as usual.
}
```

---

tar_quarto_files *Quarto file detection*

Description

Detect the important files in a Quarto project.

Usage

```
tar_quarto_files(path = ".", profile = NULL)
```

Arguments

path	Character of length 1, either the file path to a Quarto source document or the directory path to a Quarto project. Defaults to the Quarto project in the current working directory.
profile	Character of length 1, Quarto profile. If NULL, the default profile will be used. Requires Quarto version 1.2 or higher. See https://quarto.org/docs/projects/profiles.html for details.

Details

This function is just a thin wrapper that interprets the output of quarto::quarto_inspect() and returns what tarchetypes needs to know about the current Quarto project or document.

Value

A named list of important file paths in a Quarto project or document:

- sources: source files with tar_load()/tar_read() target dependencies in R code chunks.
- output: output files that will be generated during quarto::quarto_render().
- input: pre-existing files required to render the project or document, such as _quarto.yml.

See Also

Other Literate programming utilities: [tar_knitr_deps](), [tar_knitr_deps_expr]()

Examples

```
lines <- c(
  "---",
  "title: source file",
  "---",
  "Assume these lines are in report.qmd.",
  "```{r}",
  "1 + 1",
  "```"
)
path <- tempfile(fileext = ".qmd")
writeLines(lines, path)
# If Quarto is installed, run:
# tar_quarto_files(path)
```

---

tar_quarto_raw   *Target with a Quarto project (raw version).*

Description

Shorthand to include a Quarto project or standalone Quarto source document in a `targets` pipeline.

Usage

```
tar_quarto_raw(
  name,
  path = ".",
  working_directory = NULL,
  extra_files = character(0),
  execute = TRUE,
  execute_params = NULL,
  cache = NULL,
  cache_refresh = FALSE,
  debug = FALSE,
  quiet = TRUE,
  quarto_args = NULL,
```

```
    pandoc_args = NULL,
    profile = NULL,
    packages = NULL,
    library = NULL,
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = "main",
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name
: Character of length 1, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run [`tar_seed_set()`](#) on the result to locally recreate the target's initial RNG state.

path
: Character of length 1, either the single `*.qmd` source file to be rendered or a directory containing a Quarto project. Defaults to the working directory of the `targets` pipeline. Passed directly to the `input` argument of `quarto::quarto_render()`.

working_directory
: Optional character string, path to the working directory to temporarily set when running the report. The default is `NULL`, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than `NULL`, you must manually set the value of the `store` argument relative to the working directory in all calls to `tar_read()` and `tar_load()` in the report. Otherwise, these functions will not know where to find the data.

extra_files
: Character vector of extra files and directories to track for changes. The target will be invalidated (rerun on the next `tar_make()`) if the contents of these files changes. No need to include anything already in the output of [`tar_quarto_files()`](#), the list of file dependencies automatically detected through `quarto::quarto_inspect()`.

execute
: Whether to execute embedded code chunks.

execute_params
: A non-expression language object (use `quote()`, not `expression()`) that evaluates to a named list of parameters for parameterized Quarto documents. These

parameters override the custom custom elements of the `params` list in the YAML
front-matter of the Quarto source files. The list is quoted (not evaluated until the
target runs) so that upstream targets can serve as parameter values.

cache   Cache execution output (uses knitr cache and jupyter-cache respectively for Rmd
                      and Jupyter input files).

cache_refresh   Force refresh of execution cache.

debug   Leave intermediate files in place after render.

quiet   Suppress warning and other messages.

quarto_args   Character vector of other `quarto` CLI flag pass to the command. This is mainly
                      for advanced usage, e.g it can be useful for new options added to quarto CLI and
                      not yet supported as function argument.

pandoc_args   Additional command line options to pass to pandoc.

profile   Character of length 1, Quarto profile. If NULL, the default profile will be used.
                      Requires Quarto version 1.2 or higher. See https://quarto.org/docs/projects/profiles.html for details.

packages   Deprecated on 2023-09-05 (version 0.7.8.9000). Please load R packages inside
                      the Quarto report itself.

library   Deprecated on 2023-09-05 (version 0.7.8.9000). Please load R packages inside
                      the Quarto report itself.

error   Character of length 1, what to do if the target stops and throws an error. Options:

                      • "stop": the whole pipeline stops and throws an error.
                      • "continue": the whole pipeline keeps going.
                      • "abridge": any currently running targets keep running, but no new tar-
                        gets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
                      • "null": The errored target continues and returns NULL. The data hash is
                        deliberately wrong so the target is not up to date for the next run of the
                        pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                      memory until the end of the pipeline (unless `storage` is "worker", in which case
                      `targets` unloads the value from memory right after storing it in order to avoid
                      sending copious data over a network). If "transient", the target gets unloaded
                      after every new target completes. Either way, the target gets automatically
                      loaded into memory whenever another target needs the value. For cloud-based
                      dynamic files (e.g. `format = "file"` with `repository = "aws"`), this memory
                      strategy applies to the temporary local copy of the file: "persistent" means
                      it remains until the end of the pipeline and is then deleted, and "transient"
                      means it gets deleted as soon as possible. The former conserves bandwidth, and
                      the latter conserves local storage.

garbage_collection
                      Logical, whether to run `base::gc()` just before the target runs.

deployment   Character of length 1. If `deployment` is "main", then the target will run on the
                      central controlling R process. Otherwise, if `deployment` is "worker" and you
                      set up the pipeline with distributed/parallel computing, then the target runs on a

parallel worker. For more on distributed/parallel computing in `targets`, please visit <https://books.ropensci.org/targets/crew.html>.

priority | Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [`tar_make_future()`](#)).

resources | Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.

retrieval | Character of length 1, only relevant to [`tar_make_clustermq()`](#) and [`tar_make_future()`](#). Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue | An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description | Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [`tar_manifest()`](#) and [`tar_visnetwork()`](#), and they let you select subsets of targets for the `names` argument of functions like [`tar_make()`](#). For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

`tar_quarto_raw()` is just like [`tar_quarto()`](#) except that it uses standard evaluation for the `name` and `execute_params` arguments (instead of quoting them).

Value

A target object with `format = "file"`. When this target runs, it returns a sorted character vector of all the important file paths: the rendered documents, the Quarto source files, and other input and output files. The output files are determined by the YAML front-matter of standalone Quarto documents and `_quarto.yml` in Quarto projects, and you can see these files with [`tar_quarto_files()`](#) (powered by `quarto::quarto_inspect()`). All returned paths are *relative* paths to ensure portability (so that the project can be moved from one file system to another without invalidating the target). See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at <https://books.ropensci.org/targets/>. Please read the walkthrough at <https://books.ropensci.org/targets/walkthrough.html> to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Literate programming limitations

Literate programming files are messy and variable, so functions like tar_render() have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if tar_read() and/or tar_load() are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. tar_load("x") and tar_load(contains("x")) may not detect target x. * Special/optional input/output files may not be detected in all cases. * tar_render() and friends are for local files only. They do not integrate with the cloud storage capabilities of targets.

Quarto troubleshooting

If you encounter difficult errors, please read https://github.com/quarto-dev/quarto-r/issues/16. In addition, please try to reproduce the error using quarto::quarto_render("your_report.qmd", execute_dir = getwd()) without using targets at all. Isolating errors this way makes them much easier to solve.

See Also

Other Literate programming targets: tar_knit(), tar_knit_raw(), tar_quarto(), tar_quarto_rep(), tar_quarto_rep_raw(), tar_render(), tar_render_raw(), tar_render_rep(), tar_render_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({  # tar_dir() runs code from a temporary directory.
# Unparameterized Quarto document:
lines <- c(
  "---",
  "title: report.qmd source file",
  "output_format: html",
  "---",
  "Assume these lines are in report.qmd.",
  "```{r}",
  "targets::tar_read(data)",
  "```"
)
# In tar_dir(), not part of the user's file space:
writeLines(lines, "report.qmd")
# Include the report in a pipeline as follows.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tar_quarto_raw("report", path = "report.qmd")
  )
```

```
  }, ask = FALSE)
  # Then, run the pipeline as usual.

  # Parameterized Quarto:
  lines <- c(
    "---",
    "title: 'report.qmd source file with parameters'",
    "output_format: html_document",
    "params:",
    "  your_param: \"default value\"",
    "---",
    "Assume these lines are in report.qmd.",
    "```{r}",
    "print(params$your_param)",
    "```"
  )
  # In tar_dir(), not part of the user's file space:
  writeLines(lines, "report.qmd")
  # Include the report in the pipeline as follows.
  targets::tar_script({
    library(tarchetypes)
    list(
      tar_target(data, data.frame(x = seq_len(26), y = letters)),
      tar_quarto_raw(
        "report",
        path = "report.qmd",
        execute_params = quote(list(your_param = data))
      )
    )
  }, ask = FALSE)
  # Then, run the pipeline as usual.
})
}
```

---

tar_quarto_rep   *Parameterized Quarto with dynamic branching.*

Description

Targets to render a parameterized Quarto document with multiple sets of parameters.

Usage

```
tar_quarto_rep(
  name,
  path,
  working_directory = NULL,
  execute_params = data.frame(),
  batches = NULL,
  extra_files = character(0),
```

```
  execute = TRUE,
  cache = NULL,
  cache_refresh = FALSE,
  debug = FALSE,
  quiet = TRUE,
  quarto_args = NULL,
  pandoc_args = NULL,
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run tar_seed_set() on the result to locally recreate the target's initial RNG state.

path
: Character string, file path to the Quarto source file. Must have length 1.

working_directory
: Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

execute_params
: Code to generate a data frame or tibble with one row per rendered report and

one column per Quarto parameter. You may also include an `output_file` column to specify the path of each rendered report. If included, the `output_file` column must be a character vector with one and only one output file for each row of parameters. If an `output_file` column is not included, then the output files are automatically determined using the parameters, and the default file format is determined by the YAML front-matter of the Quarto source document. Only the first file format is used, the others are not generated. Quarto parameters must not be named `tar_group` or `output_file`. This `execute_params` argument is converted into the command for a target that supplies the Quarto parameters.

batches   Number of batches. This is also the number of dynamic branches created during `tar_make()`.

extra_files   Character vector of extra files that `targets` should track for changes. If the content of one of these files changes, then the report will rerun over all the parameters on the next `tar_make()`. These files are *extra* files, and they do not include the Quarto source document or rendered output document, which are already tracked for changes. Examples include bibliographies, style sheets, and supporting image files.

execute   Whether to execute embedded code chunks.

cache   Cache execution output (uses knitr cache and jupyter-cache respectively for Rmd and Jupyter input files).

cache_refresh   Force refresh of execution cache.

debug   Leave intermediate files in place after render.

quiet   Suppress warning and other messages.

quarto_args   Character vector of other `quarto` CLI flag pass to the command. This is mainly for advanced usage, e.g it can be useful for new options added to quarto CLI and not yet supported as function argument.

pandoc_args   Additional command line options to pass to pandoc.

rep_workers   Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.

tidy_eval   Logical of length 1, whether to use tidy evaluation to resolve `execute_params`. Similar to the `tidy_eval` argument of `targets::tar_target()`.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading `packages`.

format   Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

                • "vector": branching happens with `vectors::vec_slice()` and aggregation happens with `vctrs::vec_c()`.

- "list", branching happens with [[]] and aggregation happens with list().
  In the case of list iteration, tar_read(your_target) will return a list of
  lists, where the outer list has one element per batch and each inner list
  has one element per rep within batch. To un-batch this nested list, call
  tar_read(your_target, recursive = FALSE).

- "group": dplyr::group_by()-like functionality to branch over subsets of
  a data frame. The target's return value must be a data frame with a special
  tar_group column of consecutive integers from 1 through the number of
  groups. Each integer designates a group, and a branch is created for each
  collection of rows in a group. See the [tar_group()](#) function to see how
  you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new tar-
  gets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is
  deliberately wrong so the target is not up to date for the next run of the
  pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in
                 memory until the end of the pipeline (unless storage is "worker", in which case
                 targets unloads the value from memory right after storing it in order to avoid
                 sending copious data over a network). If "transient", the target gets unloaded
                 after every new target completes. Either way, the target gets automatically
                 loaded into memory whenever another target needs the value. For cloud-based
                 dynamic files (e.g. format = "file" with repository = "aws"), this memory
                 strategy applies to the temporary local copy of the file: "persistent" means
                 it remains until the end of the pipeline and is then deleted, and "transient"
                 means it gets deleted as soon as possible. The former conserves bandwidth, and
                 the latter conserves local storage.

garbage_collection
                 Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the
                 central controlling R process. Otherwise, if deployment is "worker" and you
                 set up the pipeline with distributed/parallel computing, then the target runs on a
                 parallel worker. For more on distributed/parallel computing in targets, please
                 visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first
                 when multiple competing targets are ready simultaneously. Targets with priori-
                 ties closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).

resources   Object returned by tar_resources() with optional settings for high-performance
                 computing functionality, alternative data storage formats, and other optional ca-
                 pabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#).
                 Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

`tar_quarto_rep()` is an alternative to `tar_target()` for a parameterized Quarto document that depends on other targets. Parameters must be given as a data frame with one row per rendered report and one column per parameter. An optional `output_file` column may be included to set the output file path of each rendered report. (See the `execute_params` argument for details.)

The Quarto source should mention other dependency targets tar_load() and tar_read() in the active code chunks (which also allows you to render the report outside the pipeline if the _targets/ data store already exists and appropriate defaults are specified for the parameters). (Do not use tar_load_raw() or tar_read_raw() for this.) Then, `tar_quarto()` defines a special kind of target. It 1. Finds all the tar_load()/tar_read() dependencies in the report and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use tar_load_raw() or tar_read_raw() for this.) 2. Sets format = "file" (see `tar_target()`) so `targets` watches the files at the returned paths and reruns the report if those files change. 3. Configures the target's command to return the output report files: the rendered document, the source file, and file paths mentioned in `files`. All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as deployment = "main" in the target and quiet = TRUE in quarto::quarto_render().

Value

A list of target objects to render the Quarto reports. Changes to the parameters, source file, dependencies, etc. will cause the appropriate targets to rerun during tar_make(). See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why tar_rep() and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are tar_rep2(), tar_map_rep(), tar_map2_count(), tar_map2_size(), and tar_render_rep(). For the tar_map2_*() functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed, or withr::local_seed()). For tar_render_rep(), custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with tar_quarto_rep() and the execute_params argument.

Literate programming limitations

Literate programming files are messy and variable, so functions like tar_render() have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if tar_read() and/or tar_load() are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. tar_load("x") and tar_load(contains("x")) may not detect target x. * Special/optional input/output files may not be detected in all cases. * tar_render() and friends are for local files only. They do not integrate with the cloud storage capabilities of targets.

Quarto troubleshooting

If you encounter difficult errors, please read https://github.com/quarto-dev/quarto-r/issues/16. In addition, please try to reproduce the error using quarto::quarto_render("your_report.qmd", execute_dir = getwd()) without using targets at all. Isolating errors this way makes them much easier to solve.

See Also

Other Literate programming targets: tar_knit(), tar_knit_raw(), tar_quarto(), tar_quarto_raw(), tar_quarto_rep_raw(), tar_render(), tar_render_raw(), tar_render_rep(), tar_render_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
# Parameterized Quarto:
```

```
lines <- c(
  "---",
  "title: 'report.qmd file'",
  "output_format: html_document",
  "params:",
  "  par: \"default value\"",
  "---",
  "Assume these lines are in a file called report.qmd.",
  "```{r}",
  "print(params$par)",
  "```"
)
writeLines(lines, "report.qmd") # In tar_dir(), not the user's file space.
# The following pipeline will run the report for each row of params.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_quarto_rep(
      report,
      path = "report.qmd",
      execute_params = tibble::tibble(par = c(1, 2))
    )
  )
}, ask = FALSE)
# Then, run the targets pipeline as usual.
})
}
```

---

tar_quarto_rep_raw   *Parameterized Quarto with dynamic branching (raw version).*

Description

Targets to render a parameterized Quarto document with multiple sets of parameters (raw version).
Same as tar_quarto_rep() except name is a character string, params is an expression object, and
extra arguments to quarto::quarto_render() are passed through the args argument instead of
....

Usage

```
tar_quarto_rep_raw(
  name,
  path,
  working_directory = NULL,
  execute_params = expression(NULL),
  batches = NULL,
  extra_files = character(0),
  execute = TRUE,
  cache = NULL,
```

```
    cache_refresh = FALSE,
    debug = FALSE,
    quiet = TRUE,
    quarto_args = NULL,
    pandoc_args = NULL,
    rep_workers = 1,
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = targets::tar_option_get("format"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description")
)
```

Arguments

name
Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.

path
Character string, file path to the Quarto source file. Must have length 1.

working_directory
Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

execute_params
Expression object with code to generate a data frame or tibble with one row per rendered report and one column per Quarto parameter. You may also include an output_file column to specify the path of each rendered report. If included, the output_file column must be a character vector with one and

	only one output file for each row of parameters. If an `output_file` column is not included, then the output files are automatically determined using the parameters, and the default file format is determined by the YAML front-matter of the Quarto source document. Only the first file format is used, the others are not generated. Quarto parameters must not be named `tar_group` or `output_file`. This `execute_params` argument is converted into the command for a target that supplies the Quarto parameters.
batches	Number of batches to group the Quarto files. For a large number of reports, increase the number of batches to decrease target-level overhead. Defaults to the number of reports to render (1 report per batch).
extra_files	Character vector of extra files that `targets` should track for changes. If the content of one of these files changes, then the report will rerun over all the parameters on the next `tar_make()`. These files are *extra* files, and they do not include the Quarto source document or rendered output document, which are already tracked for changes. Examples include bibliographies, style sheets, and supporting image files.
execute	Whether to execute embedded code chunks.
cache	Cache execution output (uses knitr cache and jupyter-cache respectively for Rmd and Jupyter input files).
cache_refresh	Force refresh of execution cache.
debug	Leave intermediate files in place after render.
quiet	Suppress warning and other messages.
quarto_args	Character vector of other `quarto` CLI flag pass to the command. This is mainly for advanced usage, e.g it can be useful for new options added to quarto CLI and not yet supported as function argument.
pandoc_args	Additional command line options to pass to pandoc.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Character of length 1, `format` argument to `tar_target()` to store the data frame of Quarto parameters.
iteration	Character of length 1, `iteration` argument to `tar_target()` for the Quarto documents. Does not apply to the target with Quarto parameters (whose iteration is always `"group"`).
error	Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

                Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

                - "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
                - "worker": the worker loads the targets dependencies.
                - "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_quarto_rep_raw() is an alternative to tar_target_raw() for parameterized Quarto reports that depend on other targets. Parameters must be given as a data frame with one row per rendered report and one column per parameter. An optional output_file column may be included to set the output file path of each rendered report. (See the execute_params argument for details.)

The Quarto source should mention other dependency targets tar_load() and tar_read() in the active code chunks (which also allows you to render the report outside the pipeline if the _targets/ data store already exists and appropriate defaults are specified for the parameters). (Do not use tar_load_raw() or tar_read_raw() for this.) Then, tar_quarto() defines a special kind of target. It 1. Finds all the tar_load()/tar_read() dependencies in the report and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use tar_load_raw() or tar_read_raw() for this.) 2. Sets format = "file" (see tar_target()) so targets watches the files at the returned paths and reruns the report if those files change. 3. Configures the target's command to return the output report files: the rendered document, the source file, and then the *_files/ directory if it exists. All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as deployment = "main" in the target and quiet = TRUE in quarto::quarto_render().

Value

A list of target objects to render the Quarto reports. Changes to the parameters, source file, dependencies, etc. will cause the appropriate targets to rerun during tar_make(). See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why tar_rep() and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this

seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the `tar_map2_*()` functions, it is possible to manually supply your own seeds through the `command1` argument and then invoke them in your custom code for command2 (`set.seed()`, `withr::with_seed`, or `withr::local_seed()`). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the `execute_params` argument.

Literate programming limitations

Literate programming files are messy and variable, so functions like `tar_render()` have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if `tar_read()` and/or `tar_load()` are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. `tar_load("x")` and `tar_load(contains("x"))` may not detect target x. * Special/optional input/output files may not be detected in all cases. * `tar_render()` and friends are for local files only. They do not integrate with the cloud storage capabilities of `targets`.

Quarto troubleshooting

If you encounter difficult errors, please read https://github.com/quarto-dev/quarto-r/issues/16. In addition, please try to reproduce the error using `quarto::quarto_render("your_report.qmd", execute_dir = getwd())` without using `targets` at all. Isolating errors this way makes them much easier to solve.

See Also

Other Literate programming targets: `tar_knit()`, `tar_knit_raw()`, `tar_quarto()`, `tar_quarto_raw()`, `tar_quarto_rep()`, `tar_render()`, `tar_render_raw()`, `tar_render_rep()`, `tar_render_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
# Parameterized Quarto:
lines <- c(
  "---",
  "title: 'report.qmd source file'",
  "output_format: html_document",
  "params:",
  "  par: \"default value\"",
  "---",
  "Assume these lines are in a file called report.qmd.",
  "```{r}",
  "print(params$par)",
  "```"
)
writeLines(lines, "report.qmd") # In tar_dir(), not the user's file space.
# The following pipeline will run the report for each row of params.
targets::tar_script({
  library(tarchetypes)
  list(
```

```
    tar_quarto_rep_raw(
      "report",
      path = "report.qmd",
      execute_params = quote(tibble::tibble(par = c(1, 2)))
    )
  )
}, ask = FALSE)
# Then, run the targets pipeline as usual.
})
}
```

---

tar_render   *Target with an R Markdown document.*

Description

Shorthand to include an R Markdown document in a `targets` pipeline.

Usage

```
tar_render(
  name,
  path,
  output_file = NULL,
  working_directory = NULL,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = "main",
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description"),
  quiet = TRUE,
  ...
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                bolically to induce a dependency relationship: e.g. `tar_target(downstream_target,`
                `f(upstream_target))` is a target named `downstream_target` which depends
                on a target `upstream_target` and a function `f()`. In addition, a target's name

determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](https://...) on the result to locally recreate the target's initial RNG state.

path   Character string, file path to the R Markdown source file. Must have length 1.

output_file   Character string, file path to the rendered output file.

working_directory

Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

tidy_eval   Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment	Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.
priority	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).
resources	Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.
retrieval	Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:
	• "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
	• "worker": the worker loads the targets dependencies.
	• "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.
cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".
quiet	An option to suppress printing during rendering from knitr, pandoc command line and others. To only suppress printing of the last "Output created: " message, you can set rmarkdown.render.message to FALSE
...	Named arguments to rmarkdown::render(). These arguments are evaluated when the target actually runs in tar_make(), not when the target is defined. That means, for example, you can use upstream targets as parameters of parameterized R Markdown reports. tar_render(your_target, "your_report.Rmd", params = list(your_param = your_target)) # nolint will run rmarkdown::render("your_report.Rmd" params = list(your_param = your_target)). # nolint For parameterized reports, it is recommended to supply a distinct output_file argument to each tar_render() call and set useful defaults for parameters in the R Markdown source. See the examples section for a demonstration.

Details

tar_render() is an alternative to tar_target() for R Markdown reports that depend on other targets. The R Markdown source should mention dependency targets with tar_load() and tar_read() in the active code chunks (which also allows you to render the report outside the pipeline if the _targets/ data store already exists). (Do not use tar_load_raw() or tar_read_raw() for this.)

Then, `tar_render()` defines a special kind of target. It 1. Finds all the `tar_load()`/`tar_read()` dependencies in the report and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use `tar_load_raw()` or `tar_read_raw()` for this.) 2. Sets `format = "file"` (see `tar_target()`) so `targets` watches the files at the returned paths and reruns the report if those files change. 3. Configures the target's command to return both the output report files and the input source file. All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as `deployment = "main"` in the target and `quiet = TRUE` in `rmarkdown::render()`.

Value

A target object with `format = "file"`. When this target runs, it returns a character vector of file paths: the rendered document, the source file, and then the `*_files/` directory if it exists. Unlike `rmarkdown::render()`, all returned paths are *relative* paths to ensure portability (so that the project can be moved from one file system to another without invalidating the target). See the "Target objects" section for background.

Literate programming limitations

Literate programming files are messy and variable, so functions like `tar_render()` have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if `tar_read()` and/or `tar_load()` are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. `tar_load("x")` and `tar_load(contains("x"))` may not detect target x. * Special/optional input/output files may not be detected in all cases. * `tar_render()` and friends are for local files only. They do not integrate with the cloud storage capabilities of `targets`.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Literate programming targets: `tar_knit()`, `tar_knit_raw()`, `tar_quarto()`, `tar_quarto_raw()`, `tar_quarto_rep()`, `tar_quarto_rep_raw()`, `tar_render_raw()`, `tar_render_rep()`, `tar_render_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({  # tar_dir() runs code from a temporary directory.
```

```
# Unparameterized R Markdown:
lines <- c(
  "---",
  "title: report.Rmd source file",
  "output_format: html_document",
  "---",
  "Assume these lines are in report.Rmd.",
  "```{r}",
  "targets::tar_read(data)",
  "```"
)
# Include the report in a pipeline as follows.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tar_render(report, "report.Rmd")
  )
}, ask = FALSE)
# Then, run the targets pipeline as usual.

# Parameterized R Markdown:
lines <- c(
  "---",
  "title: 'report.Rmd source file with parameters'",
  "output_format: html_document",
  "params:",
  "  your_param: \"default value\"",
  "---",
  "Assume these lines are in report.Rmd.",
  "```{r}",
  "print(params$your_param)",
  "```"
)
# Include the report in the pipeline as follows.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tar_render(report, "report.Rmd", params = list(your_param = data))
  )
}, ask = FALSE)
})
# Then, run the targets pipeline as usual.
}
```

tar_render_raw   *Target with an R Markdown document (raw version).*

Description

Shorthand to include an R Markdown document in a `targets` pipeline (raw version)

Usage

```
tar_render_raw(
  name,
  path,
  output_file = NULL,
  working_directory = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  error = targets::tar_option_get("error"),
  deployment = "main",
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description"),
  quiet = TRUE,
  render_arguments = quote(list())
)
```

Arguments

name   Character of length 1, name of the target.

path   Character string, file path to the R Markdown source file. Must have length 1.

output_file   Character string, file path to the rendered output file.

working_directory

          Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

error   Character of length 1, what to do if the target stops and throws an error. Options:

        • "stop": the whole pipeline stops and throws an error.

        • "continue": the whole pipeline keeps going.

        • "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

        • "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

quiet   An option to suppress printing during rendering from knitr, pandoc command line and others. To only suppress printing of the last "Output created: " message, you can set rmarkdown.render.message to FALSE

render_arguments
                Optional language object with a list of named arguments to rmarkdown::render(). Cannot be an expression object. (Use quote(), not expression().) The reason for quoting is that these arguments may depend on upstream targets whose values are not available at the time the target is defined, and because tar_render_raw() is the "raw" version of a function, we want to avoid all non-standard evaluation.

Details

tar_render_raw() is just like tar_render() except that it uses standard evaluation. The name argument is a character vector, and the render_arguments argument is a language object.

Value

A target object with format = "file". When this target runs, it returns a character vector of file paths: the rendered document, the source file, and then the *_files/ directory if it exists. Unlike

rmarkdown::render(), all returned paths are *relative* paths to ensure portability (so that the project can be moved from one file system to another without invalidating the target). See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Literate programming limitations

Literate programming files are messy and variable, so functions like tar_render() have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if tar_read() and/or tar_load() are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. tar_load("x") and tar_load(contains("x")) may not detect target x. * Special/optional input/output files may not be detected in all cases. * tar_render() and friends are for local files only. They do not integrate with the cloud storage capabilities of targets.

See Also

Other Literate programming targets: tar_knit(), tar_knit_raw(), tar_quarto(), tar_quarto_raw(), tar_quarto_rep(), tar_quarto_rep_raw(), tar_render(), tar_render_rep(), tar_render_rep_raw()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
# Unparameterized R Markdown report:
lines <- c(
  "---",
  "title: 'report.Rmd source file'",
  "output_format: html_document",
  "---",
  "Assume these lines are in report.Rmd.",
  "```{r}",
  "targets::tar_read(data)",
  "```"
)
# Include the report in the pipeline as follows:
targets::tar_script({
  library(tarchetypes)
  list(
    tar_target(data, data.frame(x = seq_len(26), y = letters)),
```

```
      tar_render_raw("report", "report.Rmd")
    )
}, ask = FALSE)
# Then, run the targets pipeline as usual.

# Parameterized R Markdown:
lines <- c(
  "---",
  "title: 'report.Rmd source file with parameters.'",
  "output_format: html_document",
  "params:",
  "  your_param: \"default value\"",
  "---",
  "Assume these lines are in report.Rmd.",
  "```{r}",
  "print(params$your_param)",
  "```"
)
# Include this parameterized report in the pipeline as follows.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_target(data, data.frame(x = seq_len(26), y = letters)),
    tar_render_raw(
      "report",
      "report.Rmd",
      render_arguments = quote(list(params = list(your_param = data)))
    )
  )
}, ask = FALSE)
# Then, run the targets pipeline as usual.
})
}
```

---

tar_render_rep   *Parameterized R Markdown with dynamic branching.*

Description

Targets to render a parameterized R Markdown report with multiple sets of parameters.

Usage

```
tar_render_rep(
  name,
  path,
  working_directory = NULL,
  params = data.frame(),
  batches = NULL,
  rep_workers = 1,
```

```
    packages = targets::tar_option_get("packages"),
    library = targets::tar_option_get("library"),
    format = targets::tar_option_get("format"),
    iteration = targets::tar_option_get("iteration"),
    error = targets::tar_option_get("error"),
    memory = targets::tar_option_get("memory"),
    garbage_collection = targets::tar_option_get("garbage_collection"),
    deployment = targets::tar_option_get("deployment"),
    priority = targets::tar_option_get("priority"),
    resources = targets::tar_option_get("resources"),
    retrieval = targets::tar_option_get("retrieval"),
    cue = targets::tar_option_get("cue"),
    description = targets::tar_option_get("description"),
    quiet = TRUE,
    ...
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target, f(upstream_target)) is a target named downstream_target which depends on a target upstream_target and a function f(). In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.

path
: Character string, file path to the R Markdown source file. Must have length 1.

working_directory
: Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

params
: Code to generate a data frame or tibble with one row per rendered report and one column per R Markdown parameter. You may also include an output_file column to specify the path of each rendered report. This params argument is converted into the command for a target that supplies the R Markdown parameters.

batches
: Number of batches. This is also the number of dynamic branches created during tar_make().

rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
iteration	Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vectors::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list(). In the case of list iteration, tar_read(your_target) will return a list of lists, where the outer list has one element per batch and each inner list has one element per rep within batch. To un-batch this nested list, call tar_read(your_target, recursive = FALSE).
- "group": dplyr::group_by()-like functionality to branch over subsets of a data frame. The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [tar_group()](tar_group) function to see how you can create the special tar_group column with dplyr::group_by().

error	Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory	Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

>   Logical, whether to run base::gc() just before the target runs.

deployment
>   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority
>   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources
>   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

retrieval
>   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:
>
>   - "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
>   - "worker": the worker loads the targets dependencies.
>   - "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue
>   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description
>   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

quiet
>   An option to suppress printing during rendering from knitr, pandoc command line and others. To only suppress printing of the last "Output created: " message, you can set rmarkdown.render.message to FALSE

...
>   Other named arguments to rmarkdown::render(). Unlike tar_render(), these arguments are evaluated when the target is defined, not when it is run. (The only reason to delay evaluation in tar_render() was to handle R Markdown parameters, and tar_render_rep() handles them differently.)

Details

tar_render_rep() is an alternative to tar_target() for parameterized R Markdown reports that depend on other targets. Parameters must be given as a data frame with one row per rendered report and one column per parameter. An optional output_file column may be included to set the output file path of each rendered report. The R Markdown source should mention other dependency targets tar_load() and tar_read() in the active code chunks (which also allows you to render the report outside the pipeline if the _targets/ data store already exists and appropriate defaults are specified for the parameters). (Do not use tar_load_raw() or tar_read_raw() for this.)

Then, `tar_render()` defines a special kind of target. It 1. Finds all the `tar_load()`/`tar_read()` dependencies in the report and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use `tar_load_raw()` or `tar_read_raw()` for this.) 2. Sets `format = "file"` (see `tar_target()`) so `targets` watches the files at the returned paths and reruns the report if those files change. 3. Configures the target's command to return the output report files: the rendered document, the source file, and then the `*_files/` directory if it exists. All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as `deployment = "main"` in the target and `quiet = TRUE` in `rmarkdown::render()`.

Value

A list of target objects to render the R Markdown reports. Changes to the parameters, source file, dependencies, etc. will cause the appropriate targets to rerun during `tar_make()`. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, `tar_option_get("seed")` (for `targets` version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, `tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...)` produces the same numerical output as `tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...)` (but with different batch names). Other target factories with this seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the `tar_map2_*()` functions, it is possible to manually supply your own seeds through the `command1` argument and then invoke them in your custom code for `command2` (`set.seed()`, `withr::with_seed`, or `withr::local_seed()`). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the `execute_params` argument.

Literate programming limitations

Literate programming files are messy and variable, so functions like `tar_render()` have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if `tar_read()` and/or `tar_load()` are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. `tar_load("x")` and `tar_load(contains("x"))` may not detect target x. * Special/optional input/output files may not be detected in all cases. * `tar_render()` and friends are for local files only. They do not integrate with the cloud storage capabilities of `targets`.

See Also

Other Literate programming targets: `tar_knit()`, `tar_knit_raw()`, `tar_quarto()`, `tar_quarto_raw()`, `tar_quarto_rep()`, `tar_quarto_rep_raw()`, `tar_render()`, `tar_render_raw()`, `tar_render_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
# Parameterized R Markdown:
lines <- c(
  "---",
  "title: 'report.Rmd file'",
  "output_format: html_document",
  "params:",
  "  par: \"default value\"",
  "---",
  "Assume these lines are in a file called report.Rmd.",
  "```{r}",
  "print(params$par)",
  "```"
)
# The following pipeline will run the report for each row of params.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_render_rep(
      report,
      "report.Rmd",
      params = tibble::tibble(par = c(1, 2))
    )
  )
}, ask = FALSE)
# Then, run the targets pipeline as usual.
})
}
```

---

tar_render_rep_raw   *Parameterized R Markdown with dynamic branching (raw version).*

Description

Targets to render a parameterized R Markdown report with multiple sets of parameters (raw version). Same as `tar_render_rep()` except `name` is a character string, `params` is an expression object, and extra arguments to `rmarkdown::render()` are passed through the `args` argument instead of `...`.

Usage

```
tar_render_rep_raw(
  name,
  path,
  working_directory = NULL,
  params = expression(NULL),
  batches = NULL,
  rep_workers = 1,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description"),
  quiet = TRUE,
  args = list()
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.

path
: Character string, file path to the R Markdown source file. Must have length 1.

working_directory

Optional character string, path to the working directory to temporarily set when running the report. The default is NULL, which runs the report from the current working directory at the time the pipeline is run. This default is recommended in the vast majority of cases. To use anything other than NULL, you must manually set the value of the store argument relative to the working directory in all calls to tar_read() and tar_load() in the report. Otherwise, these functions will not know where to find the data.

params   Expression object with code to generate a data frame or tibble with one row per rendered report and one column per R Markdown parameter. You may also include an output_file column to specify the path of each rendered report. R Markdown parameters must not be named tar_group or output_file. This params argument is converted into the command for a target that supplies the R Markdown parameters.

batches   Number of batches to group the R Markdown files. For a large number of reports, increase the number of batches to decrease target-level overhead. Defaults to the number of reports to render (1 report per batch).

rep_workers   Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Character of length 1, format argument to tar_target() to store the data frame of R Markdown parameters.

iteration   Character of length 1, iteration argument to tar_target() for the R Markdown documents. Does not apply to the target with R Markdown parameters (whose iteration is always "group").

error   Character of length 1, what to do if the target stops and throws an error. Options:

                • "stop": the whole pipeline stops and throws an error.
                • "continue": the whole pipeline keeps going.
                • "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
                • "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory

strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

retrieval   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

quiet   An option to suppress printing during rendering from knitr, pandoc command line and others. To only suppress printing of the last "Output created: " message, you can set rmarkdown.render.message to FALSE

args   Named list of other arguments to rmarkdown::render(). Must not include params or output_file. Evaluated when the target is defined.

Details

tar_render_rep_raw() is an alternative to tar_target_raw() for parameterized R Markdown reports that depend on other targets. Parameters must be given as a data frame with one row per rendered report and one column per parameter. An optional output_file column may be included to set the output file path of each rendered report. The R Markdown source should mention other dependency targets tar_load() and tar_read() in the active code chunks (which also allows you to

render the report outside the pipeline if the _targets/ data store already exists and appropriate defaults are specified for the parameters). (Do not use tar_load_raw() or tar_read_raw() for this.) Then, tar_render() defines a special kind of target. It 1. Finds all the tar_load()/tar_read() dependencies in the report and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use tar_load_raw() or tar_read_raw() for this.) 2. Sets format = "file" (see tar_target()) so targets watches the files at the returned paths and re-runs the report if those files change. 3. Configures the target's command to return the output report files: the rendered document, the source file, and then the *_files/ directory if it exists. All these file paths are relative paths so the project stays portable. 4. Forces the report to run in the user's current working directory instead of the working directory of the report. 5. Sets convenient default options such as deployment = "main" in the target and quiet = TRUE in rmarkdown::render().

Value

A list of target objects to render the R Markdown reports. Changes to the parameters, source file, dependencies, etc. will cause the appropriate targets to rerun during tar_make(). See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why tar_rep() and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are tar_rep2(), tar_map_rep(), tar_map2_count(), tar_map2_size(), and tar_render_rep(). For the tar_map2_*() functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed, or withr::local_seed()). For tar_render_rep(), custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with tar_quarto_rep() and the execute_params argument.

Literate programming limitations

Literate programming files are messy and variable, so functions like `tar_render()` have limitations: * Child documents are not tracked for changes. * Upstream target dependencies are not detected if `tar_read()` and/or `tar_load()` are called from a user-defined function. In addition, single target names must be mentioned and they must be symbols. `tar_load("x")` and `tar_load(contains("x"))` may not detect target x. * Special/optional input/output files may not be detected in all cases. * `tar_render()` and friends are for local files only. They do not integrate with the cloud storage capabilities of `targets`.

See Also

Other Literate programming targets: `tar_knit()`, `tar_knit_raw()`, `tar_quarto()`, `tar_quarto_raw()`, `tar_quarto_rep()`, `tar_quarto_rep_raw()`, `tar_render()`, `tar_render_raw()`, `tar_render_rep()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
# Parameterized R Markdown:
lines <- c(
  "---",
  "title: 'report.Rmd source file'",
  "output_format: html_document",
  "params:",
  "  par: \"default value\"",
  "---",
  "Assume these lines are in a file called report.Rmd.",
  "```{r}",
  "print(params$par)",
  "```"
)
# The following pipeline will run the report for each row of params.
targets::tar_script({
  library(tarchetypes)
  list(
    tar_render_rep_raw(
      "report",
      "report.Rmd",
      params = quote(tibble::tibble(par = c(1, 2)))
    )
  )
}, ask = FALSE)
# Then, run the targets pipeline as usual.
})
}
```

---

tar_rep   *Batched replication with dynamic branching.*

Description

Batching is important for optimizing the efficiency of heavily dynamically-branched workflows: https://books.ropensci.org/targets/dynamic.html#batching. tar_rep() replicates a command in strategically sized batches.

Usage

```
tar_rep(
  name,
  command,
  batches = 1,
  reps = 1,
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                bolically to induce a dependency relationship: e.g. tar_target(downstream_target,
                f(upstream_target)) is a target named downstream_target which depends
                on a target upstream_target and a function f(). In addition, a target's name
                determines its random number generator seed. In this way, each target runs with
                a reproducible seed so someone else running the same pipeline should get the
                same results, and no two targets in the same pipeline share the same seed. (Even
                dynamic branches have different names and thus different seeds.) You can re-
                cover the seed of a completed target with tar_meta(your_target, seed) and
                run tar_seed_set() on the result to locally recreate the target's initial RNG
                state.

command   R code to run multiple times. Must return a list or data frame because tar_rep()
                will try to append new elements/columns tar_batch and tar_rep to the output
                to denote the batch and rep-within-batch IDs, respectively.

batches	Number of batches. This is also the number of dynamic branches created during `tar_make()`.
reps	Number of replications in each batch. The total number of replications is `batches * reps`.
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
tidy_eval	Whether to invoke tidy evaluation (e.g. the `!!` operator from `rlang`) as soon as the target is defined (before `tar_make()`). Applies to the `command` argument.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
format	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [`tar_resources_aws()`](https://books.ropensci.org/targets/data.html), but versioning capabilities may be lost in doing so. See the cloud storage section of <https://books.ropensci.org/targets/data.html> for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of <https://books.ropensci.org/targets/data.html> for details for instructions.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

iteration	Character of length 1, name of the iteration mode of the target. Choices:

- `"vector"`: branching happens with `vectors::vec_slice()` and aggregation happens with `vctrs::vec_c()`.
- `"list"`, branching happens with `[[]]` and aggregation happens with `list()`. In the case of list iteration, `tar_read(your_target)` will return a list of lists, where the outer list has one element per batch and each inner list has one element per rep within batch. To un-batch this nested list, call `tar_read(your_target, recursive = FALSE)`.
- `"group"`: `dplyr::group_by()`-like functionality to branch over subsets of a data frame. The target's return value must be a data frame with a special `tar_group` column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [`tar_group()`](#) function to see how you can create the special `tar_group` column with `dplyr::group_by()`.

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
                         Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic

files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_rep() and tar_rep_raw() each create two targets: an upstream local stem with an integer vector of batch ids, and a downstream pattern that maps over the batch ids. (Thus, each batch is a branch.) Each batch/branch replicates the command a certain number of times. If the command returns a list or data frame, then the targets from tar_rep() will try to append new elements/columns tar_batch, tar_rep, and tar_seed to the output to denote the batch, rep-within-batch index, and rep-specific seed, respectively.

Both batches and reps within each batch are aggregated according to the method you specify in the iteration argument. If "list", reps and batches are aggregated with list(). If "vector", then vctrs::vec_c(). If "group", then vctrs::vec_rbind().

Value

A list of two targets, one upstream and one downstream. The upstream target returns a numeric index of batch ids, and the downstream one dynamically maps over the batch ids to run the command multiple times. If the command returns a list or data frame, then the targets from tar_rep() will try to append new elements/columns tar_batch and tar_rep to the output to denote the batch and rep-within-batch IDs, respectively. See the "Target objects" section for background.

tar_read(your_target) (on the downstream target with the actual work) will return a list of lists, where the outer list has one element per batch and each inner list has one element per rep within batch. To un-batch this nested list, call tar_read(your_target, recursive = FALSE).

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the tar_map2_*() functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed, or withr::local_seed()). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the execute_params argument.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`, `tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size()`, `tar_map2_size_raw()`, `tar_map_rep()`, `tar_map_rep_raw()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_rep(
      x,
      data.frame(x = sample.int(1e4, 2)),
      batches = 2,
      reps = 3
    )
  )
})
```

```
targets::tar_make()
targets::tar_read(x)
})
}
```

---

tar_rep2   *Dynamic batched computation downstream of* `tar_rep()`

Description

Batching is important for optimizing the efficiency of heavily dynamically-branched workflows: <https://books.ropensci.org/targets/dynamic.html#batching>. `tar_rep2()` uses dynamic branching to iterate over the batches and reps of existing upstream targets.

Usage

```
tar_rep2(
  name,
  command,
  ...,
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name
: Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with

a reproducible seed so someone else running the same pipeline should get the
same results, and no two targets in the same pipeline share the same seed. (Even
dynamic branches have different names and thus different seeds.) You can re-
cover the seed of a completed target with tar_meta(your_target, seed) and
run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG
state.

command	R code to run the target.
...	Symbols to name one or more upstream batched targets created by [tar_rep()](#). If you supply more than one such target, all those targets must have the same number of batches and reps per batch. And they must all return either data frames or lists. List targets must use iteration = "list" in [tar_rep()](#).
rep_workers	Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting command and pattern. If TRUE, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
format	Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a
  non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](#),
  but versioning capabilities may be lost in doing so. See the cloud stor-
  age section of https://books.ropensci.org/targets/data.html for
  details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage sec-
  tion of https://books.ropensci.org/targets/data.html for details for
  instructions.

Note: if repository is not "local" and format is "file" then the target
should create a single output file. That output file is uploaded to the cloud and
tracked for changes where it exists in the cloud. The local file is deleted after
the target runs.

iteration	Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation
  happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().

- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of [tar_target()](#) must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [tar_group()](#) function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

                 Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.

- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the names argument of functions like `tar_make()`. For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Value

A new target object to perform batched computation. See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number gen-
erator seed derived from its target name. In batched replicate, however, each batch is a target
with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate*
its own unique seed. Each replicate-specific seed is created based on the dynamic parent target
name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and
rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are in-
variant to batching structure. In other words, tar_rep(name = x, command = rnorm(1), batches =
100, reps = 1, ...) produces the same numerical output as tar_rep(name = x, command = rnorm(1),
batches = 10, reps = 10, ...) (but with different batch names). Other target factories with this
seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`.
For the tar_map2_*() functions, it is possible to manually supply your own seeds through the
command1 argument and then invoke them in your custom code for command2 (set.seed(), withr::with_seed,
or withr::local_seed()). For `tar_render_rep()`, custom seeds can be supplied to the params
argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()`
and the execute_params argument.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`,
`tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size()`, `tar_map2_size_raw()`, `tar_map_rep()`,
`tar_map_rep_raw()`, `tar_rep()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_rep(
      data1,
      data.frame(value = rnorm(1)),
      batches = 2,
      reps = 3
    ),
    tarchetypes::tar_rep(
      data2,
      list(value = rnorm(1)),
      batches = 2, reps = 3,
      iteration = "list" # List iteration is important for batched lists.
    ),
    tarchetypes::tar_rep2(
      aggregate,
      data.frame(value = data1$value + data2$value),
      data1,
      data2
    )
  )
})
targets::tar_make()
```

```
targets::tar_read(aggregate)
})
}
```

---

tar_rep_raw   *Batched replication with dynamic branching (raw version).*

Description

Batching is important for optimizing the efficiency of heavily dynamically-branched workflows:
https://books.ropensci.org/targets/dynamic.html#batching. tar_rep_raw() is just like
tar_rep() except the name is a character string and the command is a language object.

Usage

```
tar_rep_raw(
  name,
  command,
  batches = 1,
  reps = 1,
  rep_workers = 1,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Character of length 1, name of the target. A target name must be a valid name
                 for a symbol in R, and it must not start with a dot. Subsequent targets can refer to
                 this name symbolically to induce a dependency relationship: e.g. tar_target(downstream_target,
                 f(upstream_target)) is a target named downstream_target which depends
                 on a target upstream_target and a function f(). In addition, a target's name
                 determines its random number generator seed. In this way, each target runs with

a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with tar_meta(your_target, seed) and run [tar_seed_set()](#) on the result to locally recreate the target's initial RNG state.

command   Expression object with code to run multiple times. Must return a list or data frame when evaluated.

batches   Number of batches. This is also the number of dynamic branches created during tar_make().

reps   Number of replications in each batch. The total number of replications is batches * reps.

rep_workers   Positive integer of length 1, number of local R processes to use to run reps within batches in parallel. If 1, then reps are run sequentially within each batch. If greater than 1, then reps within batch are run in parallel using a PSOCK cluster.

tidy_eval   Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as the target is defined (before tar_make()). Applies to the command argument.

packages   Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use tar_option_set() to set packages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Optional storage format for the target's return value. With the exception of format = "file", each target gets a file in _targets/objects, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.

repository   Character of length 1, remote repository for target storage. Choices:

- "local": file system of the local machine.
- "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of [tar_resources_aws()](#), but versioning capabilities may be lost in doing so. See the cloud storage section of <https://books.ropensci.org/targets/data.html> for details for instructions.
- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of <https://books.ropensci.org/targets/data.html> for details for instructions.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().

- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of [tar_target()](#) must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [tar_group()](#) function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

              Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.

- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_rep_raw() creates two targets: an upstream local stem with an integer vector of batch ids, and a downstream pattern that maps over the batch ids. (Thus, each batch is a branch.) Each batch/branch replicates the command a certain number of times.

Both batches and reps within each batch are aggregated according to the method you specify in the iteration argument. If "list", reps and batches are aggregated with list(). If "vector", then vctrs::vec_c(). If "group", then vctrs::vec_rbind().

Value

A list of two target objects, one upstream and one downstream. The upstream one does some work and returns some file paths, and the downstream target is a pattern that applies format = "file". See the "Target objects" section for background.

Replicate-specific seeds

In ordinary pipelines, each target has its own unique deterministic pseudo-random number generator seed derived from its target name. In batched replicate, however, each batch is a target with multiple replicate within that batch. That is why `tar_rep()` and friends give each *replicate* its own unique seed. Each replicate-specific seed is created based on the dynamic parent target name, tar_option_get("seed") (for targets version 0.13.5.9000 and above), batch index, and rep-within-batch index. The seed is set just before the replicate runs. Replicate-specific seeds are invariant to batching structure. In other words, `tar_rep(name = x, command = rnorm(1), batches = 100, reps = 1, ...)` produces the same numerical output as `tar_rep(name = x, command = rnorm(1), batches = 10, reps = 10, ...)` (but with different batch names). Other target factories with this seed scheme are `tar_rep2()`, `tar_map_rep()`, `tar_map2_count()`, `tar_map2_size()`, and `tar_render_rep()`. For the `tar_map2_*()` functions, it is possible to manually supply your own seeds through the command1 argument and then invoke them in your custom code for command2 (`set.seed()`, `withr::with_seed`, or `withr::local_seed()`). For `tar_render_rep()`, custom seeds can be supplied to the params argument and then invoked in the individual R Markdown reports. Likewise with `tar_quarto_rep()` and the execute_params argument.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other branching: `tar_combine()`, `tar_combine_raw()`, `tar_map()`, `tar_map2()`, `tar_map2_count()`, `tar_map2_count_raw()`, `tar_map2_raw()`, `tar_map2_size()`, `tar_map2_size_raw()`, `tar_map_rep()`, `tar_map_rep_raw()`, `tar_rep()`, `tar_rep2()`, `tar_rep2_raw()`, `tar_rep_map()`, `tar_rep_map_raw()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_rep_raw(
      "x",
      expression(data.frame(x = sample.int(1e4, 2))),
      batches = 2,
      reps = 3
    )
  )
})
```

```
targets::tar_make(callr_function = NULL)
targets::tar_read(x)
})
}
```

---

tar_select_names   *Select target names from a target list*

Description

Select the names of targets from a target list.

Usage

```
tar_select_names(targets, ...)
```

Arguments

targets   A list of target objects as described in the "Target objects" section. It does not
             matter how nested the list is as long as the only leaf nodes are targets.

...   One or more comma-separated `tidyselect` expressions, e.g. `starts_with("prefix")`.
             Just like `...` in `dplyr::select()`.

Value

A character vector of target names.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists
of target objects. Target objects represent skippable steps of the analysis pipeline as described
at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.
ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis
pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories
explains target factories (functions like this one which generate targets) and the design specifica-
tion at https://books.ropensci.org/targets-design/ details the structure and composition
of target objects.

See Also

Other target selection: `tar_select_targets()`

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets <- list(
  list(
    targets::tar_target(x, 1),
    targets::tar_target(y1, 2)
  ),
  targets::tar_target(y2, 3),
  targets::tar_target(z, 4)
)
tar_select_names(targets, starts_with("y"), contains("z"))
})
}
```

---

tar_select_targets   *Select target objects from a target list*

Description

Select target objects from a target list.

Usage

```
tar_select_targets(targets, ...)
```

Arguments

targets	A list of target objects as described in the "Target objects" section. It does not matter how nested the list is as long as the only leaf nodes are targets.
...	One or more comma-separated tidyselect expressions, e.g. starts_with("prefix"). Just like ... in dplyr::select().

Value

A list of target objects. See the "Target objects" section of this help file.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other target selection: [tar_select_names](){.underline}()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets <- list(
  list(
    targets::tar_target(x, 1),
    targets::tar_target(y1, 2)
  ),
  targets::tar_target(y2, 3),
  targets::tar_target(z, 4)
)
tar_select_targets(targets, starts_with("y"), contains("z"))
})
}
```

---

tar_skip   *Target with a custom cancellation condition.*

Description

Create a target that cancels itself if a user-defined decision rule is met.

Usage

```
tar_skip(
  name,
  command,
  skip,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name   Symbol, name of the target. A target name must be a valid name for a symbol in
                R, and it must not start with a dot. Subsequent targets can refer to this name sym-
                bolically to induce a dependency relationship: e.g. tar_target(downstream_target,
                f(upstream_target)) is a target named downstream_target which depends
                on a target upstream_target and a function f(). In addition, a target's name
                determines its random number generator seed. In this way, each target runs with
                a reproducible seed so someone else running the same pipeline should get the
                same results, and no two targets in the same pipeline share the same seed. (Even
                dynamic branches have different names and thus different seeds.) You can re-
                cover the seed of a completed target with tar_meta(your_target, seed) and
                run tar_seed_set() on the result to locally recreate the target's initial RNG
                state.

command   R code to run the target.

skip   R code for the skipping condition. If it evaluates to TRUE during tar_make(),
                the target will cancel itself.

tidy_eval   Whether to invoke tidy evaluation (e.g. the !! operator from rlang) as soon as
                the target is defined (before tar_make()). Applies to arguments command and
                skip.

packages   Character vector of packages to load right before the target runs or the output
                data is reloaded for downstream targets. Use tar_option_set() to set pack-
                ages globally for all subsequent targets you define.

library   Character vector of library paths to try when loading packages.

format   Optional storage format for the target's return value. With the exception of
                format = "file", each target gets a file in _targets/objects, and each format
                is a different way to save and load this file. See the "Storage formats" section
                for a detailed list of possible data storage formats.

repository   Character of length 1, remote repository for target storage. Choices:

                • "local": file system of the local machine.
                • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a
                  non-AWS S3 bucket using the endpoint argument of tar_resources_aws(),
                  but versioning capabilities may be lost in doing so. See the cloud stor-
                  age section of https://books.ropensci.org/targets/data.html for
                  details for instructions.
                • "gcp": Google Cloud Platform storage bucket. See the cloud storage sec-
                  tion of https://books.ropensci.org/targets/data.html for details for
                  instructions.

                Note: if repository is not "local" and format is "file" then the target
                should create a single output file. That output file is uploaded to the cloud and
                tracked for changes where it exists in the cloud. The local file is deleted after
                the target runs.

iteration   Character of length 1, name of the iteration mode of the target. Choices:

                • "vector": branching happens with vctrs::vec_slice() and aggregation
                  happens with vctrs::vec_c().

- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of [tar_target()](#) must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the [tar_group()](#) function to see how you can create the special tar_group column with dplyr::group_by().

error   Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

memory   Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network). If "transient", the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical, whether to run base::gc() just before the target runs.

deployment   Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority   Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in [tar_make_future()](#)).

resources   Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.

- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

  If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

  The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, storage = "none" is completely unnecessary if format is "file".

retrieval   Character of length 1, only relevant to [tar_make_clustermq()](#) and [tar_make_future()](#). Must be one of the following values:

- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "worker": the worker loads the targets dependencies.
- "none": the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

cue   An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

description   Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like [tar_manifest()](#) and [tar_visnetwork()](#), and they let you select subsets of targets for the names argument of functions like [tar_make()](#). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

tar_skip() creates a target that cancels itself whenever a custom condition is met. The mechanism of cancellation is targets::tar_cancel(your_condition), which allows skipping to happen even if the target does not exist yet. This behavior differs from tar_cue(mode = "never"), which still runs if the target does not exist.

Value

A target object with targets::tar_cancel(your_condition) inserted into the command. See the "Target objects" section for background.

Target objects

Most tarchetypes functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described

at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other targets with custom invalidation rules: tar_change(), tar_download(), tar_force()

Examples

```
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
targets::tar_dir({ # tar_dir() runs code from a temporary directory.
targets::tar_script({
  list(
    tarchetypes::tar_skip(x, command = "value", skip = 1 > 0)
  )
})
targets::tar_make()
})
}
```

---

tar_sub   *Create multiple expressions with symbol substitution.*

Description

Loop over a grid of values and create an expression object from each one. Helps with general metaprogramming.

Usage

```
tar_sub(expr, values)
```

Arguments

expr   Starting expression. Values are iteratively substituted in place of symbols in
            expr to create each new expression.

values   List of values to substitute into expr to create the expressions. All elements of
            values must have the same length.

Value

A list of expression objects. Often, these expression objects evaluate to target objects (but not necessarily). See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Metaprogramming utilities: `tar_eval()`, `tar_eval_raw()`, `tar_sub_raw()`

Examples

```
# tar_map() is incompatible with tar_render() because the latter
# operates on preexisting tar_target() objects. By contrast,
# tar_eval() and tar_sub() iterate over code farther upstream.
values <- list(
  name = lapply(c("name1", "name2"), as.symbol),
  file = list("file1.Rmd", "file2.Rmd")
)
tar_sub(tar_render(name, file), values = values)
```

---

tar_sub_raw *Create multiple expressions with symbol substitution (raw version).*

Description

Loop over a grid of values and create an expression object from each one. Helps with general metaprogramming. Unlike `tar_sub()`, which quotes the expr argument, tar_sub_raw() assumes expr is an expression object.

Usage

```
tar_sub_raw(expr, values)
```

Arguments

expr   Expression object with the starting expression. Values are iteratively substituted in place of symbols in expr to create each new expression.

values   List of values to substitute into expr to create the expressions. All elements of values must have the same length.

Value

A list of expression objects. Often, these expression objects evaluate to target objects (but not necessarily). See the "Target objects" section for background.

Target objects

Most `tarchetypes` functions are target factories, which means they return target objects or lists of target objects. Target objects represent skippable steps of the analysis pipeline as described at https://books.ropensci.org/targets/. Please read the walkthrough at https://books.ropensci.org/targets/walkthrough.html to understand the role of target objects in analysis pipelines.

For developers, https://wlandau.github.io/targetopia/contributing.html#target-factories explains target factories (functions like this one which generate targets) and the design specification at https://books.ropensci.org/targets-design/ details the structure and composition of target objects.

See Also

Other Metaprogramming utilities: `tar_eval()`, `tar_eval_raw()`, `tar_sub()`

Examples

```
# tar_map() is incompatible with tar_render() because the latter
# operates on preexisting tar_target() objects. By contrast,
# tar_eval_raw() and tar_sub_raw() iterate over code farther upstream.
values <- list(
  name = lapply(c("name1", "name2"), as.symbol),
  file = c("file1.Rmd", "file2.Rmd")
)
tar_sub_raw(quote(tar_render(name, file)), values = values)
```

Index