Package ‘tarchetypes’

August 4, 2021

Title Archetypes for Targets

Description Function-oriented Make-like declarative workflows 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 workflows 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 0.3.0

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 digest (>= 0.6.25), fs (>= 1.4.2), rlang (>= 0.4.7), targets (>= 0.6.0), tibble (>= 3.0.1), tidyselect (>= 1.1.0), utils, vctrs (>= 0.3.4), withr (>= 2.1.2)

Suggests curl (>= 4.3), dplyr (>= 1.0.0), knitr (>= 1.28), rmarkdown (>= 2.1), testthat (>= 3.0.0), xml2 (>= 1.3.2)

Encoding UTF-8

Language en-US

Config/testthat/edition 3

RoxygenNote 7.1.1

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@gmail.com>
Repository  CRAN
Date/Publication  2021-08-04 04:40:02 UTC

R topics documented:

- tarchetypes-package .................................................. 3
- tar_age ................................................................. 3
- tar_change .............................................................. 6
- tar_combine .............................................................. 9
- tar_combine_raw ....................................................... 12
- tar_cue_age ............................................................. 15
- tar_cue_age_raw ....................................................... 17
- tar_cue_force .......................................................... 19
- tar_cue_skip ........................................................... 21
- tar_download ............................................................. 23
- tar_eval ................................................................. 26
- tar_eval_raw ............................................................ 27
- tar_files ................................................................. 29
- tar_files_input ......................................................... 32
- tar_files_input_raw .................................................... 35
- tar_files_raw ........................................................... 38
- tar_force ................................................................. 41
- tar_formats ............................................................... 44
- tar_group_by ............................................................ 55
- tar_group_count ......................................................... 58
- tar_group_select ........................................................ 60
- tar_group_size ........................................................... 63
- tar_hook_before .......................................................... 66
- tar_hook_inner ........................................................... 68
- tar_hook_outer ........................................................... 69
- tar_knit ................................................................. 71
- tar_knitr_deps ........................................................... 74
- tar_knitr_deps_expr ..................................................... 75
- tar_knit_raw ............................................................. 76
- tar_map ................................................................. 78
- tar_plan ................................................................. 80
- tar_render ............................................................... 81
- tar_render_raw ......................................................... 84
- tar_render_rep .......................................................... 87
- tar_render_rep_raw ...................................................... 91
- tar_rep ................................................................. 94
- tar_rep_map ............................................................. 98
- tar_rep_map_raw ......................................................... 101
- tar_rep_raw ............................................................ 105
- tar_select_names ....................................................... 108
- tar_select_targets ...................................................... 109
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.

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"),
  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")
)
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 may 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 builds. 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()`.

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.)

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 such as `format = "aws_file"`, this memory strategy applies to temporary local copies of the file in `_targets/scratch/`: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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.
**tar_age**

**deployment** Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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()`. If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

**retrieval** Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

**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()`.

**Details**

`tar_age()` uses the cue from `tar_cue_age()`, which 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 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 combine with targets::tar_invalidate() right before calling `tar_make()`. For example, `tar_invalidate(all_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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](https://books.ropensci.org/targets-design/) details the structure and composition of target objects.

**See Also**

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

**Examples**

```r
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**

```r
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"),
  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")
)

Arguments

name
Symbol, name of the target. 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 set.seed() on the result to locally recreate the target's initial RNG state.

command
R code to run the target.

dependence
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 builds. 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 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 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.)

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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_clustermq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built 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(). If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval
Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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.

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.
**tar_combine**

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"),
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")
)

**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 builds. 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 `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 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.)

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 unload 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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_cluster mq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built 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_cluster mq() and tar_make_future(). If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval
Character of length 1, only relevant to tar_make_cluster mq() and tar_make_future(). If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

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_rep_map_raw()`, `tar_rep_map()`, `tar_rep()`, `tar_rep()`

Examples

```r
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 <- tararchetypes::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

```r
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"),
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")
)

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 builds. 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 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 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.)

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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_clustermq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built 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(). If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

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_rep_map_raw(), tar_rep_map(), tar_rep_raw(), tar_rep()

Examples

```r
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. In other words, if the target produces any data (as opposed to tracking input files or URLs) then the target will rerun periodically at regular intervals of time.

Usage

tar_cue_age(
  name,
  age,
  command = TRUE,
  depend = TRUE,
  format = 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()`.
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 combine with `targets::tar_invalidate()` right before calling `tar_make()`. For example, `tar_invalidate(all_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

```r
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. In other words, if the target produces any data (as opposed to tracking input files or URLs) then the target will rerun periodically at regular intervals of time.

Usage

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Character of length 1, name of the target.</td>
</tr>
<tr>
<td>age</td>
<td>A difftime object of length 1, such as as.difftime(3, units = &quot;days&quot;). 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 = &quot;days&quot;).</td>
</tr>
<tr>
<td>command</td>
<td>Logical, whether to rerun the target if command changed since last time.</td>
</tr>
<tr>
<td>depend</td>
<td>Logical, whether to rerun the target if the value of one of the dependencies changed.</td>
</tr>
<tr>
<td>format</td>
<td>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().</td>
</tr>
<tr>
<td>iteration</td>
<td>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().</td>
</tr>
<tr>
<td>file</td>
<td>Logical, whether to rerun the target if the file(s) with the return value changed or at least one is missing.</td>
</tr>
</tbody>
</table>

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 combine with targets::tar_invalidate() right before calling tar_make(). For example, tar_invalidate(all_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: \texttt{tar\_age()}, \texttt{tar\_cue\_age()}, \texttt{tar\_cue\_force()}, \texttt{tar\_cue\_skip()}

Examples

```r
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 \hspace{1cm} \textit{Cue to force a target to run if a condition is true}

Description

\texttt{tar\_cue\_force()} creates a cue object to force a target to run if an arbitrary condition evaluates to \texttt{TRUE}. Supply the returned cue object to the cue argument of \texttt{targets::tar\_target()} or similar.

Usage

```r
tar\_cue\_force(
  condition,
  command = TRUE,
  depend = TRUE,
  format = 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()`.
- **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_raw()`, `tar_cue_age()`, `tar_cue_skip()`

Examples

```r
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    targets::tar_script({
      library(tarchetypes)
    })
  })
}```
```r
tar_cue_skip

list(
  targets::tar_target(
    data,
    data.frame(x = seq_len(26)),
    cue = tarchetypes::tar_cue_force(1 > 0)
  )
)
}

Order

Tarfile targets

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

```r
tar_cue_skip(
  condition,
  command = TRUE,
  depend = TRUE,
  format = 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()`.
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 = tarchetypes::tar_cue_force())` 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_age()`, `tar_cue_force()`

Examples

```r
if (identical(Sys.getenv("TAR_LONG_EXAMPLES"), "true")) {
  targets::tar_dir({
    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_script(
    library(tarchetypes)
    list(
      targets::tar_target(
        data,
        data.frame(x = seq_len(26)), # Change the command.
        cue = tarchetypes::tar_cue_force() 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

```r
 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")
)
```

Arguments

- **name** Symbol, name of the target. 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 `set.seed()` 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.
Character vector of local file paths to download each of the URLs. Must be known and declared before the pipeline runs.

Method to be used for downloading files. Current download methods are "internal", "wininet" (Windows only) "libcurl", "wget" and "curl", and there is a value "auto": see ‘Details’ and ‘Note’.

The method can also be set through the option "download.file.method": see options().

If TRUE, suppress status messages (if any), and the progress bar.

Character of length 1, name of the iteration mode of the target. Choices:
- "vector": branching happens with \texttt{vctrs::vec\_slice()} and aggregation happens with \texttt{vctrs::vec\_c()}.
- "list": branching happens with \texttt{[[\]]} and aggregation happens with \texttt{list()}.
- "group": \texttt{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 \texttt{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 \texttt{tar\_group()} function to see how you can create the special \texttt{tar\_group} column with \texttt{dplyr::group\_by()}.

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 \url{https://books.ropensci.org/targets/debugging.html} to learn how to debug targets using saved workspaces.)

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 such as format = "aws\_file", this memory strategy applies to temporary local copies of the file in \_targets/\_scratch/"; "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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()`. If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

**retrieval**
Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

**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 automatically 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

```r
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

```r
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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](https://books.ropensci.org/targets-design/) details the structure and composition of target objects.

See Also

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

Examples

```r
# 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(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_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(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  

Easy dynamic branching over files or urls.

Description

Shorthand for a pattern that correctly branches over files or urls.

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", "url", "aws_file"),
  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")
)

Arguments

name  Symbol, name of the target. Subsequent targets can refer to this name symboli-
cally 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 some-
one 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 com-
pleted target with tar_meta(your_target,seed) and run set.seed() 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 builds. 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.

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 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.)

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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_clustermq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built earlier (and polled earlier in tar_make_future()).
**tar_files**

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()`. If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval  
Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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.

**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 = "url"`, or `format = "aws_file"`. 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_input_raw()`, `tar_files_input()`, `tar_files_raw()`
### Examples

```r
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)
}
```

### Description

Easy dynamic branching over known existing input files or urls.

### Usage

```r
tar_files_input(
  name,
  files,
  batches = length(files),
  format = c("file", "url", "aws_file"),
  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")
)
```

### Arguments

- **name**
  Symbol, name of the target. 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 \texttt{tar\_meta(your\_target,seed)} and run \texttt{set.seed()} on the result to locally recreate the target's initial RNG state.

\textbf{files}  
Nonempty character vector of known existing input files to track for changes.

\textbf{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.

\textbf{format}  
Character, either "file" or "url". See the format argument of \texttt{targets::tar\_target()} for details.

\textbf{iteration}  
Character, iteration method. Must be a method supported by the iteration argument of \texttt{targets::tar\_target()}. The iteration method for the upstream target is always "list" in order to support batching.

\textbf{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 \url{https://books.ropensci.org/targets/debugging.html} to learn how to debug targets using saved workspaces.)

\textbf{memory}  
Character of length 1, memory strategy. If "persistent", the target stays in memory until the end of the pipeline (unless \texttt{storage} is "worker", in which case \texttt{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 such as \texttt{format = "aws\_file"}, this memory strategy applies to temporary local copies of the file in _targets/scratch/_: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

\textbf{garbage\_collection}  
Logical, whether to run \texttt{base::gc()} just before the target runs.

\textbf{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 built earlier (and polled earlier in \texttt{tar\_make\_future()}).

\textbf{resources}  
Object returned by \texttt{tar\_resources()} with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of \texttt{targets}. See \texttt{tar\_resources()} for details.

\textbf{cue}  
An optional object from \texttt{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.
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 = "url"`, or `format = "aws_file"`. 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_input_raw()`, `tar_files_raw()`, `tar_files()`

Examples

```r
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,
```
Easy dynamic branching over known existing files or urls (raw version).

Description
Shorthand for a pattern that correctly branches over files or urls.

Usage
tar_files_input_raw(
  name,
  files,
  batches = length(files),
  format = c("file", "url", "aws_file"),
  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")
)

Arguments

name Symbol, name of the target. 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 set.seed() 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.
**tar_files_input_raw**

- **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" or "url". See the format argument of targets::tar_target() for details.

- **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](https://books.ropensci.org/targets/debugging.html) to learn how to debug targets using saved workspaces.)

- **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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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 built 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.

**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", format = "url", or format = "aws_file". 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_input(), tar_files_raw(), tar_files()

Examples

```r
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)
}}
```
Easy dynamic branching over files or urls (raw version).

Description

Shorthand for a pattern that correctly branches over files or urls.

Usage

tar_files_raw(
  name,
  command,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  format = c("file", "url", "aws_file"),
  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")
)

Arguments

name Symbol, name of the target. 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 `set.seed()` 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 builds. 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.
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 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.)

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 such as `format = "aws_file"`, this memory strategy applies to temporary local copies of the file in `_targets/scratch/`: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built 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()`. If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval

Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.
```r
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)
      )
    }
  })
```

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.

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 = "url", or format = "aws_file". 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_input_raw()`, `tar_files_input()`, `tar_files()`

Examples
```r
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)
      )
    })
  })
```
\texttt{tar\_force}

\begin{verbatim}
})
targets::tar_make()
targets::tar_read(x)
})
}
\end{verbatim}

\begin{itemize}
\item \textbf{Description}
\begin{itemize}
\item Create a target that always runs if a user-defined condition rule is met.
\end{itemize}
\item \textbf{Usage}
\begin{verbatim}
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"),
    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")
)
\end{verbatim}
\item \textbf{Arguments}
\begin{itemize}
\item \texttt{name} Symbol, name of the target. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. \texttt{tar\_target(downstream\_target,f(upstream\_target))} is a target named downstream\_target which depends on a target upstream\_target and a function \texttt{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 \texttt{tar\_meta(your\_target,seed)} and run \texttt{set.seed()} on the result to locally recreate the target's initial RNG state.
\end{itemize}
\end{itemize}
**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 builds. 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 `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 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](https://books.ropensci.org/targets/debugging.html) to learn how to debug targets using saved workspaces.)

**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 such as `format = "aws_file"`, this memory strategy applies to temporary local copies of the file in `_targets/scratch/"; "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_clusterMQ()` and `tar_make_future()`.

If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.
tar_force

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>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 built earlier (and polled earlier in tar_make_future()).</td>
</tr>
<tr>
<td>resources</td>
<td>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.</td>
</tr>
<tr>
<td>storage</td>
<td>Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If &quot;main&quot;, the target’s return value is sent back to the host machine and saved locally. If &quot;worker&quot;, the worker saves the value.</td>
</tr>
<tr>
<td>retrieval</td>
<td>Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If &quot;main&quot;, the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If &quot;worker&quot;, the worker loads the targets dependencies.</td>
</tr>
<tr>
<td>cue</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```r
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**

```r
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"),
  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")
)
```

**tar_file**

```r
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"),
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")
)


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"),
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")
)

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"),
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")
)

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"),
  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")
)

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"),
  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")
)

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"),
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")
)

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"),
    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")
)

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"),
    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")
)
tar_formats

) 

) 

) 

)
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")
}

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"),
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")
)

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"),
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")
)
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"),
    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")
)

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"),
    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")
)

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"),
    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")
)

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"),
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")
)

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"),
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")
)
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"),
    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")
)

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"),
    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")
)

Arguments

name Symbol, name of the target. 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 `set.seed()` on the
result to locally recreate the target’s initial RNG state.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>R code to run the target.</td>
</tr>
<tr>
<td>pattern</td>
<td>Language to define branching for a target. For example, in a pipeline with</td>
</tr>
<tr>
<td></td>
<td>numeric vector targets $x$ and $y$, <code>tar_target(z, x + y, pattern = map(x, y))</code></td>
</tr>
<tr>
<td></td>
<td>implicitly defines branches of $z$ that each compute $x[1] + y[1], x[2] + y[2]$,</td>
</tr>
<tr>
<td></td>
<td>and so on. See the user manual for details.</td>
</tr>
<tr>
<td>tidy_eval</td>
<td>Logical, whether to enable tidy evaluation when interpreting <code>command</code> and</td>
</tr>
<tr>
<td></td>
<td><code>pattern</code>. If <code>TRUE</code>, you can use the &quot;bang-bang&quot; operator <code>!!</code> to</td>
</tr>
<tr>
<td></td>
<td>programmatically insert the values of global objects.</td>
</tr>
<tr>
<td>packages</td>
<td>Character vector of packages to load right before the target builds. Use</td>
</tr>
<tr>
<td></td>
<td><code>tar_option_set()</code> to set packages globally for all subsequent targets you</td>
</tr>
<tr>
<td></td>
<td>define.</td>
</tr>
<tr>
<td>library</td>
<td>Character vector of library paths to try when loading packages.</td>
</tr>
<tr>
<td>iteration</td>
<td>Character of length 1, name of the iteration mode of the target. Choices:</td>
</tr>
<tr>
<td></td>
<td>• &quot;vector&quot;: branching happens with <code>vctrs::vec_slice()</code> and aggregation</td>
</tr>
<tr>
<td></td>
<td>happens with <code>vctrs::vec_c()</code>.</td>
</tr>
<tr>
<td></td>
<td>• &quot;list&quot;, branching happens with <code>[[[]]</code> and aggregation happens with <code>list()</code></td>
</tr>
<tr>
<td></td>
<td>• &quot;group&quot;: <code>dplyr::group_by()</code>-like functionality to branch over subsets of</td>
</tr>
<tr>
<td></td>
<td>a data frame. The target’s return value must be a data frame with a special</td>
</tr>
<tr>
<td></td>
<td><code>tar_group</code> column of consecutive integers from 1 through the number of</td>
</tr>
<tr>
<td></td>
<td>groups. Each integer designates a group, and a branch is created for each</td>
</tr>
<tr>
<td></td>
<td>collection of rows in a group. See the <code>tar_group()</code> function to see how</td>
</tr>
<tr>
<td></td>
<td>you can create the special <code>tar_group</code> column with <code>dplyr::group_by()</code>.</td>
</tr>
<tr>
<td>error</td>
<td>Character of length 1, what to do if the target stops and throws an error.</td>
</tr>
<tr>
<td></td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td>• &quot;stop&quot;: the whole pipeline stops and throws an error.</td>
</tr>
<tr>
<td></td>
<td>• &quot;continue&quot;: the whole pipeline keeps going.</td>
</tr>
<tr>
<td></td>
<td>• &quot;abridge&quot;: any currently running targets keep running, but no new targets</td>
</tr>
<tr>
<td></td>
<td>launch after that. (Visit <a href="https://books.ropensci.org/targets/debugging.html">https://books.ropensci.org/targets/debugging.html</a> to learn how to debug targets using saved workspaces.)</td>
</tr>
<tr>
<td>memory</td>
<td>Character of length 1, memory strategy. If &quot;persistent&quot;, the target stays</td>
</tr>
<tr>
<td></td>
<td>in memory until the end of the pipeline (unless storage is &quot;worker&quot;, in which</td>
</tr>
<tr>
<td></td>
<td>case targets unloads the value from memory right after storing it in order</td>
</tr>
<tr>
<td></td>
<td>to avoid sending copious data over a network). If &quot;transient&quot;, the target</td>
</tr>
<tr>
<td></td>
<td>gets unloaded after every new target completes. Either way, the target gets</td>
</tr>
<tr>
<td></td>
<td>automatically loaded into memory whenever another target needs the value. For</td>
</tr>
<tr>
<td></td>
<td>cloud-based dynamic files such as <code>format = &quot;aws_file&quot;</code>, this memory strategy</td>
</tr>
<tr>
<td></td>
<td>applies to temporary local copies of the file in `_targets/scratch/&quot;: &quot;persistent&quot;</td>
</tr>
<tr>
<td></td>
<td>means they remain until the end of the pipeline, and &quot;transient&quot; means they</td>
</tr>
<tr>
<td></td>
<td>get deleted from the file system as soon as possible. The former conserves</td>
</tr>
<tr>
<td></td>
<td>bandwidth, and the latter conserves local storage.</td>
</tr>
<tr>
<td>garbage_collection</td>
<td>Logical, whether to run <code>base::gc()</code> just before the target runs.</td>
</tr>
<tr>
<td>deployment</td>
<td>Character of length 1, only relevant to <code>tar_make_cluster()</code> and <code>tar_make_future()</code>.</td>
</tr>
<tr>
<td></td>
<td>If &quot;worker&quot;, the target builds on a parallel worker. If &quot;main&quot;, the target</td>
</tr>
<tr>
<td></td>
<td>builds on the host machine / process managing the pipeline.</td>
</tr>
</tbody>
</table>
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 built 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()`. If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

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

```r
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)
      )
    )
  }
}


**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**

```r
tar_group_by(
  name,  
  command,
  ...
)

```  

**Arguments**

- **name**
  
  Symbol, name of the target. 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 set.seed() 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 builds. 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.

**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.)

**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 such as `format = "aws_file"`, this memory strategy applies to temporary local copies of the file in `_targets/scratch/`: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make clustermq()` and `tar_make future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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()`. If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

**retrieval**
Character of length 1, only relevant to `tar_make clustermq()` and `tar_make future()`. If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.
An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

**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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](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**

```r
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"),
  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")
)

Arguments

name  Symbol, name of the target. 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 set.seed() on the result to locally recreate the target’s initial RNG state.

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 builds. 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.

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.)

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 such as `format = "aws_file"`, this memory strategy applies to temporary local copies of the file in `_targets/scratch/"; "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built 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()`. If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval

Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

cue

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

**Examples**

```r
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)
  })
}
**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**

```r
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"),
  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")
)
```

**Arguments**

- **name**: Symbol, name of the target. 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 `set.seed()` 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 builds. 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.
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.)
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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_clustermq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.
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 built 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(). If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.
retrieval Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.
cue An optional object from tar_cue() to customize the rules that decide whether the target is up to date.

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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](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**

```r
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"),
  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")
)

Arguments

name  Symbol, name of the target. 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 set.seed() 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 builds. 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.

tidy_eval  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.)

**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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_clustermq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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(). If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

**retrieval**

Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

**cue**

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

**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)`

**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.
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.

Name of targets in the target list to apply the hook. You can supply symbols, a character vector, or tidyselect helpers like `starts_with()`. Targets not included in `names` still remain in the target list, but they are not modified because the hook does not apply to them.

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.

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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](https://books.ropensci.org/targets-design/) details the structure and composition of target objects.

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

```r
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)
}
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)

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. You can supply symbols, a character vector, or tidyselect helpers like starts_with(). 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. You can supply symbols, a character vector, or tidyselect helpers like starts_with().

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

```r
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)
```
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. You can supply symbols, a character vector, or tidyselect helpers like `starts_with()`. Targets not included in `names` still remain in the target list, but they are not modified because the hook does not apply to them.

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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](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

```r
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,
  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"),
  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"),
  quiet = TRUE,
  ...
)

Arguments

name

Symbol, name of the target. 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 `set.seed()` 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.

**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 builds. 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](https://books.ropensci.org/targets/debugging.html) to learn how to debug targets using saved workspaces.)

**deployment**
Character of length 1, only relevant to `tar_make_cluster()` and `tar_make_future()`.
If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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_cluster()` and `tar_make_future()`.
If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

**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_render_raw(), tar_render_rep_raw(), tar_render_rep(), tar_render()

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()
    })
  })
}
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()

Examples

lines <- c(
  "---",
  "title: report",
  "output_format: html_document",
  "---",
  "`
  "targets::tar_load(data1)"
  "targets::tar_read(data2)"
  "```"
) report <- tempfile()
writeLines(lines, report)
tar_knitr_deps(report)
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()`

Examples

```r
lines <- c(
  "---",
  "title: report",
  "output_format: html_document",
  "---",
  "targets::tar_load(data1)",
  "targets::tar_read(data2)",
  "\`
)
report <- tempfile()
writeLines(lines, report)
tar_knitr_deps_expr(report)
```
Description

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

Usage

tar_knit_raw(
    name,
    path,
    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"),
    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.
packages Character vector of packages to load right before the target builds. 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.)
deployment Character of length 1, only relevant to tar_make_cluster() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.
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 built 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()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

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 values 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 argument 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 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()`, `tar_render_raw()`, `tar_render_rep_raw()`, `tar_render_rep()`, `tar_render()`
Examples

```r
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(), 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. Elements of the values list should be small objects that can easily deparse to names, such as characters, integers, and symbols. For more complicated elements of values, such as lists with multiple numeric vectors, `tar_map()` attempts to parse the elements into expressions, but this process is not perfect, and the default target names come out garbled. To create a list of symbols as a column of values, use `rlang::syms()`.
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. You can supply symbols, a character vector, or tidyselect helpers like 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_raw(), tar_combine(), tar_rep_map_raw(), tar_rep_map(), tar_rep_raw(), tar_rep()

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))
      )
    )
  )
}
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

```r
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.
            )
        })
    }
}
```

**tar_render**

*Target with an R Markdown document.*

**Description**

Shorthand to include an R Markdown document in a targets pipeline.

**Usage**

```r
tar_render(
    name,
    path,
    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"),
    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"),
    quiet = TRUE,
    ...
)
```

**Arguments**

- **name** Symbol, name of the target. 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 \texttt{tar\_meta(your\_target,seed)} and run \texttt{set.seed()} on the result to locally recreate the target's initial RNG state.

\textbf{path} \hspace{1cm} Character string, file path to the R Markdown source file. Must have length 1.

\textbf{tidy\_eval} \hspace{1cm} Logical, whether to enable tidy evaluation when interpreting command and pattern. If \texttt{TRUE}, you can use the "bang-bang" operator !! to programmatically insert the values of global objects.

\textbf{packages} \hspace{1cm} Character vector of packages to load right before the target builds. Use \texttt{tar\_option\_set()} to set packages globally for all subsequent targets you define.

\textbf{library} \hspace{1cm} Character vector of library paths to try when loading packages.

\textbf{error} \hspace{1cm} 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 \url{https://books.ropensci.org/targets/debugging.html} to learn how to debug targets using saved workspaces.)

\textbf{deployment} \hspace{1cm} Character of length 1, only relevant to \texttt{tar\_make\_clustermq()} and \texttt{tar\_make\_future()}. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

\textbf{priority} \hspace{1cm} 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 built earlier (and polled earlier in \texttt{tar\_make\_future()}).

\textbf{resources} \hspace{1cm} Object returned by \texttt{tar\_resources()} with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See \texttt{tar\_resources()} for details.

\textbf{retrieval} \hspace{1cm} Character of length 1, only relevant to \texttt{tar\_make\_clustermq()} and \texttt{tar\_make\_future()}. If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

\textbf{cue} \hspace{1cm} An optional object from \texttt{tar\_cue()} to customize the rules that decide whether the target is up to date.

\textbf{quiet} \hspace{1cm} 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 \texttt{rmarkdown.render.message} to \texttt{FALSE}.

... \hspace{1cm} Named arguments to \texttt{rmarkdown::render()}. These arguments are evaluated when the target actually runs in \texttt{tar\_make()}, not when the target is defined. That means, for example, you can use upstream targets as parameters of parameterized R Markdown reports. \texttt{tar\_render(your\_target, "your\_report.Rmd", params = list(your\_param = your\_target))} \# nolint will run \texttt{rmarkdown::render("your\_report.Rmd", params = list(your\_param = your\_target))}. \# nolint For parameterized reports, it is recommended to supply a distinct \texttt{output\_file} argument to each \texttt{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.

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_knit(), tar_render_raw(), tar_render_rep_raw(), tar_render_rep()

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.
```
library(tarchetypes)
list(
  tar_target(data, data.frame(x = seq_len(26), y = letters)),
  tar_render(report, "report.Rmd")
)
```

# 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.

---

**Description**

Shorthand to include an R Markdown document in a targets pipeline (raw version).
Usage

```r
tar_render_raw(
  name,  
  path, 
  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"), 
  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.
- **packages**: Character vector of packages to load right before the target builds. 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.)
- **deployment**: Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.
- **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 built 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()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.
- **cue**: An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
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.

See Also

Other Literate programming targets: tar_knit_raw(), tar_knit(), tar_render_rep_raw(), tar_render_rep(), tar_render()

Examples

```r
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.
})
})

### Description

Targets to render a parameterized R Markdown report with multiple sets of parameters.
Usage

tar_render_rep(
  name,
  path,
  params = data.frame(),
  batches = NULL,
  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"),
  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"),
  quiet = TRUE,
  ...
)

Arguments

name          Symbol, name of the target. 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 set.seed() 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.

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.

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).

packages      Character vector of packages to load right before the target builds. 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.
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 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.)

deployment

character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

priority

decimal 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 built 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()`. If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

cue

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

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.

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.

See Also

Other Literate programming targets: tar_knit_raw(), tar_knit(), tar_render_raw(), tar_render_rep_raw(), tar_render()

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.
})
}

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,
  params = expression(NULL),
  batches = NULL,
  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"),
  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"),
  quiet = TRUE,
  args = list()
)
Arguments

**name**
Symbol, name of the target. 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 `set.seed()` 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.

**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`.

**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).

**packages**
Character vector of packages to load right before the target builds. 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](https://books.ropensci.org/targets/debugging.html) to learn how to debug targets using saved workspaces.)

**deployment**
Character of length 1, only relevant to `tar_make_cluster mq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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_cluster mq()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to
the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

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.
See Also

Other Literate programming targets: `tar_knit_raw()`, `tar_knit()`, `tar_render_raw()`, `tar_render_rep()`, `tar_render()`

Examples

```r
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.
  })
}
```

---

<table>
<thead>
<tr>
<th><code>tar_rep</code></th>
<th>Batched replication with dynamic branching.</th>
</tr>
</thead>
</table>

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

```r
 tar_rep(
   name,
```
command,
batches = 1,
reps = 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"),
itration = 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")
)

Arguments

name    Symbol, name of the target. 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 `set.seed()` 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`.

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 builds. 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 targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to tar_make_cluster() and tar_make_future().
  If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built earlier (and polled earlier in tar_make_future()).

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

storage Character of length 1, only relevant to tar_make_cluster() and tar_make_future().
  If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.
Character of length 1, only relevant to \texttt{tar\_make\_clustermq()} and \texttt{tar\_make\_future}(). If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

\texttt{cue} \hspace{2cm} An optional object from \texttt{tar\_cue()} to customize the rules that decide whether the target is up to date.

\section*{Details}

\texttt{tar\_rep()} and \texttt{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 \texttt{tar\_rep()} will try to append new elements/columns \texttt{tar\_batch} and \texttt{tar\_rep} to the output to denote the batch and rep-within-batch IDs, 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 \texttt{list()}. If "vector", then \texttt{vctrs::vec\_c()}. If "group", then \texttt{vctrs::vec\_rbind()}.

\section*{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 \texttt{tar\_rep()} will try to append new elements/columns \texttt{tar\_batch} and \texttt{tar\_rep} to the output to denote the batch and rep-within-batch IDs, respectively. See the "Target objects" section for background.

\texttt{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 \texttt{tar\_read(your\_target,recursive = FALSE)}.

\section*{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 \url{https://books.ropensci.org/targets/}. Please read the walkthrough at \url{https://books.ropensci.org/targets/walkthrough.html} to understand the role of target objects in analysis pipelines.

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

\section*{See Also}

Other branching: \texttt{tar\_combine\_raw()}, \texttt{tar\_combine()}, \texttt{tar\_map()}, \texttt{tar\_rep\_map\_raw()}, \texttt{tar\_rep\_map()}, \texttt{tar\_rep\_raw()}
Examples

```r
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, # tar_rep() repeatedly creates a temporary directory.
          data.frame(x = sample.int(1e4, 2)),
          batches = 2,
          reps = 3
        )
      )
    )
  })
  targets::tar_make()
  targets::tar_read(x)
}
```

**tar_rep_map**

_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](https://books.ropensci.org/targets/dynamic.html#batching). **tar_rep_map()** uses dynamic branching to iterate over the batches and reps of existing upstream targets.

**Usage**

```r
tar_rep_map(
  name, # The name of the target.
  command, # The command to be executed.
  ..., # Additional arguments.
  tidy_eval = targets::tar_option_get("tidy_eval"), # Execute commands in a tidy eval context.
  packages = targets::tar_option_get("packages"), # Important packages to be loaded.
  library = targets::tar_option_get("library"), # Library to be loaded.
  format = targets::tar_option_get("format"), # The format of the output.
  iteration = targets::tar_option_get("iteration"), # The iteration number.
  error = targets::tar_option_get("error"), # Error handling.
  memory = targets::tar_option_get("memory"), # Memory allocation.
  garbage_collection = targets::tar_option_get("garbage_collection"), # Garbage collection.
  deployment = targets::tar_option_get("deployment"), # Deployment details.
  priority = targets::tar_option_get("priority"), # Priority.
  resources = targets::tar_option_get("resources"), # Resource allocation.
  storage = targets::tar_option_get("storage"), # Storage details.
  retrieval = targets::tar_option_get("retrieval"), # Retrieval details.
  cue = targets::tar_option_get("cue")
)
```
Arguments

name | Symbol, name of the target. 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 set.seed() 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().

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 builds. 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 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 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.)

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 such as `format = "aws_file"`, this memory strategy applies to temporary local copies of the file in `_targets/scratch/`: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_clustermsg()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built 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_clustermsg()` and `tar_make_future()`. If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

retrieval
Character of length 1, only relevant to `tar_make_clustermsg()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

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.
See Also

Other branching: `tar_combine_raw()`, `tar_combine()`, `tar_map()`, `tar_rep_map_raw()`, `tar_rep_raw()`, `tar_rep()`

Examples

```r
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_rep_map(
          aggregate,
          data.frame(value = data1$value + data2$value),
          data1,
          data2
        )
      )
    })
  })
  targets::tar_make()
  targets::tar_read(aggregate)
}
```

---

**tar_rep_map_raw**

*Batched computation downstream of tar_rep() (raw version)*

**Description**

Batching is important for optimizing the efficiency of heavily dynamically-branched workflows: [https://books.ropensci.org/targets/dynamic.html#batching](https://books.ropensci.org/targets/dynamic.html#batching). `tar_rep_map_raw()` is just like `tar_rep_map()` except it accepts a character of length 1 for name, a language object for command, and a character vector of the names of the upstream batched targets.

**Usage**

```r
tar_rep_map_raw(
  name,
```
command,
targets,
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"),
storage = targets::tar_option_get("storage"),
retrieval = targets::tar_option_get("retrieval"),
cue = targets::tar_option_get("cue")
)

Arguments

name Symbol, name of the target. 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 set.seed() on the result to locally recreate the target’s initial RNG state.

command R code to run the target.

targets Character vector of names of 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().

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 builds. 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 `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 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](https://books.ropensci.org/targets/debugging.html) to learn how to debug targets using saved workspaces.)

**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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/_: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

**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 built 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()`. If "main", the target's return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

**retrieval**

Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If "main", the target's dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

**cue**

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

A new target object to perform batched computation downstream of `tar_rep()`. 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_combine()`, `tar_map()`, `tar_rep_map()`, `tar_rep_raw()`, `tar_rep()`

Examples

```r
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_rep_map_raw("aggregate", quote(data.frame(value = data1$value + data2$value)), targets = c("data1", "data2"))
      )
    })
  }
  targets::tar_make()
  targets::tar_read(aggregate)
}
```
**Description**

Batching is important for optimizing the efficiency of heavily dynamically-branched workflows: [https://books.ropensci.org/targets/dynamic.html#batching](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**

```r
tar_rep_raw(
  name,  
  command,  
  batches = 1,  
  reps = 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"),  
  storage = targets::tar_option_get("storage"),  
  retrieval = targets::tar_option_get("retrieval"),  
  cue = targets::tar_option_get("cue")
)
```

**Arguments**

- `name` Character of length 1, name of the target. 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 `set.seed()` 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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
| 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 builds. 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 `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 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.) |
| 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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in `_targets/scratch/"`: "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system 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, only relevant to `tar_make_cluster mq()` and `tar_make_future()`. If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline. |
**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 built 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_clustermp()` and `tar_make_future()`. If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

**retrieval**  Character of length 1, only relevant to `tar_make_clustermp()` and `tar_make_future()`. If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

### 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.

### 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_combine()`, `tar_map()`, `tar_rep_map_raw()`, `tar_rep_map()`, `tar_rep()`
Examples

```r
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

```r
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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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_targets()`

Examples

```r
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**

```r
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()

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

```r
  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"),
    ... )
```
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")
)

Arguments

name
Symbol, name of the target. 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 set.seed() 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 builds. 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 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 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().
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](https://books.ropensci.org/targets/debugging.html) to learn how to debug targets using saved workspaces.)

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 such as format = "aws_file", this memory strategy applies to temporary local copies of the file in _targets/scratch/": "persistent" means they remain until the end of the pipeline, and "transient" means they get deleted from the file system as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

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

Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "worker", the target builds on a parallel worker. If "main", the target builds on the host machine / process managing the pipeline.

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 built earlier (and polled earlier in tar_make_future()).

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.

Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "main", the target’s return value is sent back to the host machine and saved locally. If "worker", the worker saves the value.

Character of length 1, only relevant to tar_make_clustermq() and tar_make_future(). If "main", the target’s dependencies are loaded on the host machine and sent to the worker before the target builds. If "worker", the worker loads the targets dependencies.

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

`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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](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**

```r
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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](https://books.ropensci.org/targets-design/) details the structure and composition of target objects.

See Also

Other Metaprogramming utilities: `tar_eval_raw()`, `tar_eval()`, `tar_sub_raw()`

Examples

```r
# 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)
```

**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 `tar_chetypes` 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/](https://books.ropensci.org/targets/). Please read the walkthrough at [https://books.ropensci.org/targets/walkthrough.html](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](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/](https://books.ropensci.org/targets-design/) details the structure and composition of target objects.

See Also

Other Metaprogramming utilities: `tar_eval_raw()`, `tar_eval()`, `tar_sub()`

Examples

```r
# 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

* Dynamic branching over files
  tar_files, 29
  tar_files_input, 32
  tar_files_input_raw, 35
  tar_files_raw, 38

* Formats
  tar_formats, 44

* Grouped data frame targets
  tar_group_by, 55
  tar_group_count, 58
  tar_group_select, 60
  tar_group_size, 63

* Literate programming targets
  tar_knit, 71
  tar_knit_raw, 76
  tar_render, 81
  tar_render_raw, 84
  tar_render_rep, 87
  tar_render_rep_raw, 91

* Literate programming utilities
  tar_knitr_deps, 74
  tar_knitr_deps_expr, 75

* Metaprogramming utilities
  tar_eval, 26
  tar_eval_raw, 27
  tar_sub, 113
  tar_sub_raw, 114

* Pipeline factories
  tar_plan, 80

* branching
  tar_combine, 9
  tar_combine_raw, 12
  tar_map, 78
  tar_rep, 94
  tar_rep_map, 98
  tar_rep_map_raw, 101
  tar_rep_raw, 105

* cues
  tar_age, 3
  tar_cue_age, 15
  tar_cue_age_raw, 17
  tar_cue_force, 19
  tar_cue_skip, 21

* hooks
  tar_hook_before, 66
  tar_hook_inner, 68
  tar_hook_outer, 69

* target selection
  tar_select_names, 108
  tar_select_targets, 109

* targets with custom invalidation rules
  tar_change, 6
  tar_download, 23
  tar_force, 41
  tar_skip, 110

options, 24
starts_with(), 67, 68, 70, 79

  tar_age, 3, 17, 19, 20, 22
  tar_aws_file (tar_formats), 44
  tar_aws_fst (tar_formats), 44
  tar_aws_fst_dt (tar_formats), 44
  tar_aws_fst_tbl (tar_formats), 44
  tar_aws_keras (tar_formats), 44
  tar_aws_parquet (tar_formats), 44
  tar_aws_qs (tar_formats), 44
  tar_aws_rds (tar_formats), 44
  tar_aws_torch (tar_formats), 44
  tar_change, 6, 25, 43, 113
  tar_change(), 43
  tar_combine, 9, 15, 79, 97, 101, 104, 107
  tar_combine(), 12
  tar_combine_raw, 12, 12, 79, 97, 101, 104, 107
  tar_cue_age, 6, 15, 19, 20, 22
  tar_cue_age(), 5
  tar_cue_age_raw, 6, 17, 17, 20, 22