Package ‘gittargets’

December 4, 2023

Title Data Version Control for the Targets Package

Description In computationally demanding data analysis pipelines, the 'targets' R package (2021, <doi:10.21105/joss.02959>) maintains an up-to-date set of results while skipping tasks that do not need to rerun. This process increases speed and increases trust in the final end product. However, it also overwrites old output with new output, and past results disappear by default. To preserve historical output, the 'gittargets' package captures version-controlled snapshots of the data store, and each snapshot links to the underlying commit of the source code. That way, when the user rolls back the code to a previous branch or commit, 'gittargets' can recover the data contemporaneous with that commit so that all targets remain up to date.

Version 0.0.7

License MIT + file LICENSE

URL `https://docs.ropensci.org/gittargets/`, `https://github.com/ropensci/gittargets`

BugReports `https://github.com/ropensci/gittargets/issues`

Depends R (>= 3.5.0)

Imports callr (>= 3.0.0), cli (>= 3.1.0), data.table (>= 1.12.8), gert (>= 1.0.0), processx (>= 3.0.0), stats, targets (>= 0.6.0), tibble (>= 3.0.0), utils, uuid (>= 0.1.4)

Suggests knitr (>= 1.30), markdown (>= 1.1), rmarkdown (>= 2.4), testthat (>= 3.0.0)

SystemRequirements Git (>= 2.0.0)

Encoding UTF-8

Language en-US

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.2.3

NeedsCompilation no

Author William Michael Landau [aut, cre]
(<https://orcid.org/0000-0003-1878-3253>),
Saras Windecker [rev],
David Neuzerling [rev],
Eli Lilly and Company [cph]

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

Repository CRAN

Date/Publication 2023-12-04 21:50:02 UTC

R topics documented:

`gittargets-package` *targets: Dynamic Function-Oriented Make-Like Declarative Pipelines for R*

Description

In computationally demanding data analysis pipelines, the `targets` R package maintains an up-to-date set of results while skipping tasks that do not need to rerun. This process increases speed and increases trust in the final end product. However, it also overwrites old output with new output, and past results disappear by default. To preserve historical output, the `gittargets` package captures version-controlled snapshots of the data store, and each snapshot links to the underlying commit of the source code. That way, when the user rolls back the code to a previous branch or commit, `gittargets` can recover the data contemporaneous with that commit so that all targets remain up to date.

`tar_git_checkout` *Check out a snapshot of the data (Git)*

Description

Check out a snapshot of the data associated with a particular code commit (default: `HEAD`).

Usage

```
tar_git_checkout(
  ref = "HEAD",
  code = getwd(),
  store = targets::tar_config_get("store"),
  force = FALSE,
  verbose = TRUE
)
```

Arguments

`ref`	Character of length 1. SHA1 hash, branch name, or other reference in the code repository that points to a code commit. (You can also identify the code commit by supplying a data branch of the form `code=<SHA1>`.) Defaults to `"HEAD"`, which points to the currently checked out code commit. Once the desired code commit is identified, `tar_git_snapshot()` checks out the latest corresponding data snapshot. There may be earlier data snapshots corresponding to this code commit, but `tar_git_snapshot()` only checks out the latest one. To check out an earlier superseded data snapshot, you will need to manually use command line Git in the data repository. If `tar_git_snapshot()` cannot find a data snapshot for the desired code commit, then it will throw an error. For a list of commits in the current code branch that have available data snapshots, see the `commit_code` column of the output of `tar_git_log()`.
`code`	Character of length 1, directory path to the code repository, usually the root of the `targets` project.
`store`	Character of length 1, path to the data store of the pipeline. If `NULL`, the `store` setting is left unchanged in the YAML configuration file (default: `_targets.yaml`). Usually, the data store lives at `_targets`. Set `store` to a custom directory to specify a path other than `_targets/`. The path need not exist before the pipeline begins, and it need not end with "_targets", but it must be writeable. For optimal performance, choose a storage location with fast read/write access. If the argument `NULL`, the setting is not modified. Use `tar_config_unset()` to delete a setting.
`force`	ignore conflicts and overwrite modified files
`verbose`	Logical of length 1, whether to print R console messages confirming that a snapshot was created.

Value

Nothing (invisibly).

See Also

Other git: `tar_git_init()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's filespace.
# Work on an initial branch.
targets::tar_script(tar_target(data, "old_data"))
targets::tar_make()
targets::tar_read(data) # "old_data"
gert::git_init()
gert::git_add("_targets.R")
gert::git_commit("First commit")
gert::git_branch_create("old_branch")
tar_git_init()
# Work on a new branch.
tar_git_snapshot(status = FALSE, verbose = FALSE)
targets::tar_script(tar_target(data, "new_data"))
targets::tar_make()
targets::tar_read(data) # "new_data"
gert::git_branch_create("new_branch")
gert::git_add("_targets.R")
gert::git_commit("Second commit")
tar_git_snapshot(status = FALSE, verbose = FALSE)
# Go back to the old branch.
gert::git_branch_checkout("old_branch")
# The target is out of date because we only reverted the code.
targets::tar_outdated()
# But tar_git_checkout() lets us restore the old version of the data!
tar_git_checkout()
targets::tar_read(data) # "old_data"
# Now, the target is up to date! And we did not even have to rerun it!
targets::tar_outdated()
})
}
```

`tar_git_init` *Initialize a data repository (Git).*

Description

Initialize a Git repository for a `targets` data store.

Usage

```
tar_git_init(
  store = targets::tar_config_get("store"),
  stash_gitignore = TRUE,
  git_lfs = TRUE,
  verbose = TRUE
)
```

Arguments

`store`	Character of length 1, path to the data store of the pipeline. If `NULL`, the `store` setting is left unchanged in the YAML configuration file (default: `_targets.yaml`). Usually, the data store lives at `_targets`. Set `store` to a custom directory to specify a path other than `_targets/`. The path need not exist before the pipeline begins, and it need not end with "_targets", but it must be writeable. For optimal performance, choose a storage location with fast read/write access. If the argument `NULL`, the setting is not modified. Use `tar_config_unset()` to delete a setting.
`stash_gitignore`	Logical of length 1, whether to temporarily stash the `.gitignore` file of the data store. See the "Stashing .gitignore" section for details.
`git_lfs`	Logical, whether to automatically opt into Git LFS to track large files in `_targets/objects` more efficiently. If `TRUE` and Git LFS is installed, it should work automatically. If `FALSE`, you can always opt in later by running `git lfs track objects` inside the data store.
`verbose`	Logical of length 1, whether to print messages to the R console.

Details

`tar_git_init()` also writes a `.gitattributes` file to the store to automatically track target output date with `git-lfs` if it is installed on your system.

Value

`NULL` (invisibly).

Stashing .gitignore

The `targets` package writes a `.gitignore` file to new data stores in order to prevent accidental commits to the code Git repository. Unfortunately, for `gittargets`, this automatic `.gitignore` file interferes with proper data versioning. So by default, `gittargets` temporarily stashes it to a hidden file called `.gittargets_gitignore` inside the data store. If your R program crashes while the stash is active, you can simply move it manually back to `.gitignore` or run `tar_git_status_data()` to restore the stash automatically if no `.gitignore` already exists.

See Also

Other git: `tar_git_checkout()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's file space.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
tar_git_init()
})
}
```

`tar_git_log` *Data snapshots of a code branch (Git)*

Description

Show all the data snapshots of a code branch.

Usage

```
tar_git_log(
  code = getwd(),
  store = targets::tar_config_get("store"),
  branch = gert::git_branch(repo = code),
  max = 100
)
```

Arguments

`code`	Character of length 1, directory path to the code repository, usually the root of the `targets` project.
`store`	Character of length 1, path to the data store of the pipeline. If `NULL`, the `store` setting is left unchanged in the YAML configuration file (default: `_targets.yaml`). Usually, the data store lives at `_targets`. Set `store` to a custom directory to specify a path other than `_targets/`. The path need not exist before the pipeline begins, and it need not end with "_targets", but it must be writeable. For optimal performance, choose a storage location with fast read/write access. If the argument `NULL`, the setting is not modified. Use `tar_config_unset()` to delete a setting.
`branch`	Character of length 1, name of the code repository branch to query. Defaults to the currently checked-out code branch.
`max`	Positive numeric of length 1, maximum number of code commits to inspect for the given branch.

Details

By design, `tar_git_log()` only queries a single code branch at a time. This allows `tar_git_log()` to report more detailed information about the snapshots of the given code branch. To query all data snapshots over all branches, simply run `gert::git_branch_list(local = TRUE, repo = "_targets")`. The valid snapshots show `"code=<SHA1>"` in the `name` column, where `<SHA1>` is the Git commit hash of the code commit corresponding to the data snapshot.

Value

A data frame of information about data snapshots and code commits.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's filespace.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
gert::git_init()
gert::git_add("_targets.R")
gert::git_commit("First commit")
tar_git_init()
tar_git_snapshot(status = FALSE, verbose = FALSE)
tar_git_log()
})
}
```

`tar_git_ok` *Check Git*

Description

Check if Git is installed and if `user.name` and `user.email` are configured globally.

Usage

```
tar_git_ok(verbose = TRUE)
```

Arguments

`verbose`	Whether to print messages to the console.

Details

You can install Git from https://git-scm.com/downloads/ and configure your identity using the instructions at https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup. You may find it convenient to run `gert::git_config_global()` with `name` equal to `user.name` and `user.email`.

Value

Logical of length 1, whether Git is installed and configured correctly.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_log()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
tar_git_ok()
```

`tar_git_snapshot` *Snapshot the data repository (Git).*

Description

Snapshot the Git data repository of a `targets` project.

Usage

```
tar_git_snapshot(
  message = NULL,
  ref = "HEAD",
  code = getwd(),
  script = targets::tar_config_get("script"),
  store = targets::tar_config_get("store"),
  stash_gitignore = TRUE,
  reporter = targets::tar_config_get("reporter_outdated"),
  envir = parent.frame(),
  callr_function = callr::r,
  callr_arguments = NULL,
  status = interactive(),
  force = FALSE,
  pack_refs = TRUE,
  verbose = TRUE
)
```

Arguments

`message`	Optional Git commit message of the data snapshot. If `NULL`, then the message is the Git commit message of the matching code commit.
`ref`	Character of length 1, reference (branch name, Git SHA1 hash, etc.) of the code commit that will map to the new data snapshot. Defaults to the commit checked out right now.
`code`	Character of length 1, directory path to the code repository, usually the root of the `targets` project.
`script`	Character of length 1, path to the target script file. Defaults to `tar_config_get("script")`, which in turn defaults to `_targets.R`. When you set this argument, the value of `tar_config_get("script")` is temporarily changed for the current function call. See `tar_script()`, `tar_config_get()`, and `tar_config_set()` for details about the target script file and how to set it persistently for a project.
`store`	Character of length 1, path to the data store of the pipeline. If `NULL`, the `store` setting is left unchanged in the YAML configuration file (default: `_targets.yaml`). Usually, the data store lives at `_targets`. Set `store` to a custom directory to specify a path other than `_targets/`. The path need not exist before the pipeline begins, and it need not end with "_targets", but it must be writeable. For optimal performance, choose a storage location with fast read/write access. If the argument `NULL`, the setting is not modified. Use `tar_config_unset()` to delete a setting.
`stash_gitignore`	Logical of length 1, whether to temporarily stash the `.gitignore` file of the data store. See the "Stashing .gitignore" section for details.
`reporter`	Character of length 1, name of the reporter to user. Controls how messages are printed as targets are checked. Choices: • `"silent"`: print nothing. • `"forecast"`: print running totals of the checked and outdated targets found so far.
`envir`	An environment, where to run the target R script (default: `_targets.R`) if `callr_function` is `NULL`. Ignored if `callr_function` is anything other than `NULL`. `callr_function` should only be `NULL` for debugging and testing purposes, not for serious runs of a pipeline, etc. The `envir` argument of `tar_make()` and related functions always overrides the current value of `tar_option_get("envir")` in the current R session just before running the target script file, so whenever you need to set an alternative `envir`, you should always set it with `tar_option_set()` from within the target script file. In other words, if you call `tar_option_set(envir = envir1)` in an interactive session and then `tar_make(envir = envir2, callr_function = NULL)`, then `envir2` will be used.
`callr_function`	A function from `callr` to start a fresh clean R process to do the work. Set to `NULL` to run in the current session instead of an external process (but restart your R session just before you do in order to clear debris out of the global environment). `callr_function` needs to be `NULL` for interactive debugging, e.g. `tar_option_set(debug = "your_target")`. However, `callr_function` should not be `NULL` for serious reproducible work.

`callr_arguments`	A list of arguments to `callr_function`.
`status`	Logical of length 1, whether to print the project status with `tar_git_status()` and ask whether a snapshot should be created.
`force`	Logical of length 1. Force checkout the data branch of an existing data snapshot of the current code commit?
`pack_refs`	Logical of length 1, whether to run `git pack-refs --all` in the data store after taking the snapshot. Packing references improves efficiency when the number of snapshots is large. Learn more at `https://git-scm.com/docs/git-pack-refs`.
`verbose`	Logical of length 1, whether to print R console messages confirming that a snapshot was created.

Details

A Git-backed `gittargets` data snapshot is a special kind of Git commit. Every data commit is part of a branch specific to the current code commit. That way, when you switch branches or commits in the code, `tar_git_checkout()` checks out the latest data snapshot that matches the code in your workspace. That way, your targets can stay up to date even as you transition among multiple branches.

Stashing .gitignore

The `targets` package writes a `.gitignore` file to new data stores in order to prevent accidental commits to the code Git repository. Unfortunately, for `gittargets`, this automatic `.gitignore` file interferes with proper data versioning. So by default, `gittargets` temporarily stashes it to a hidden file called `.gittargets_gitignore` inside the data store. If your R program crashes while the stash is active, you can simply move it manually back to `.gitignore` or run `tar_git_status_data()` to restore the stash automatically if no `.gitignore` already exists.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's filespace.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
gert::git_init()
gert::git_add("_targets.R")
gert::git_commit("First commit")
tar_git_init()
tar_git_snapshot(status = FALSE)
})
}
```

tar_git_status *Status of the project (Git)*

Description

Print the status of the code repository, the data repository, and the targets.

Usage

```
tar_git_status(
  code = getwd(),
  script = targets::tar_config_get("script"),
  store = targets::tar_config_get("store"),
  stash_gitignore = TRUE,
  reporter = targets::tar_config_get("reporter_outdated"),
  envir = parent.frame(),
  callr_function = callr::r,
  callr_arguments = NULL
)
```

Arguments

`code`	Character of length 1, directory path to the code repository, usually the root of the `targets` project.
`script`	Character of length 1, path to the target script file. Defaults to `tar_config_get("script")`, which in turn defaults to `_targets.R`. When you set this argument, the value of `tar_config_get("script")` is temporarily changed for the current function call. See `tar_script()`, `tar_config_get()`, and `tar_config_set()` for details about the target script file and how to set it persistently for a project.
`store`	Character of length 1, path to the data store of the pipeline. If `NULL`, the `store` setting is left unchanged in the YAML configuration file (default: `_targets.yaml`). Usually, the data store lives at `_targets`. Set `store` to a custom directory to specify a path other than `_targets/`. The path need not exist before the pipeline begins, and it need not end with "_targets", but it must be writeable. For optimal performance, choose a storage location with fast read/write access. If the argument `NULL`, the setting is not modified. Use `tar_config_unset()` to delete a setting.
`stash_gitignore`	Logical of length 1, whether to temporarily stash the `.gitignore` file of the data store. See the "Stashing .gitignore" section for details.
`reporter`	Character of length 1, name of the reporter to user. Controls how messages are printed as targets are checked. Choices: • `"silent"`: print nothing. • `"forecast"`: print running totals of the checked and outdated targets found so far.

`envir`	An environment, where to run the target R script (default: `_targets.R`) if `callr_function` is `NULL`. Ignored if `callr_function` is anything other than `NULL`. `callr_function` should only be `NULL` for debugging and testing purposes, not for serious runs of a pipeline, etc. The `envir` argument of `tar_make()` and related functions always overrides the current value of `tar_option_get("envir")` in the current R session just before running the target script file, so whenever you need to set an alternative `envir`, you should always set it with `tar_option_set()` from within the target script file. In other words, if you call `tar_option_set(envir = envir1)` in an interactive session and then `tar_make(envir = envir2, callr_function = NULL)`, then `envir2` will be used.
`callr_function`	A function from `callr` to start a fresh clean R process to do the work. Set to `NULL` to run in the current session instead of an external process (but restart your R session just before you do in order to clear debris out of the global environment). `callr_function` needs to be `NULL` for interactive debugging, e.g. `tar_option_set(debug = "your_target")`. However, `callr_function` should not be `NULL` for serious reproducible work.
`callr_arguments`	A list of arguments to `callr_function`.

Value

`NULL` (invisibly). Status information is printed to the R console.

Stashing .gitignore

The `targets` package writes a `.gitignore` file to new data stores in order to prevent accidental commits to the code Git repository. Unfortunately, for `gittargets`, this automatic `.gitignore` file interferes with proper data versioning. So by default, `gittargets` temporarily stashes it to a hidden file called `.gittargets_gitignore` inside the data store. If your R program crashes while the stash is active, you can simply move it manually back to `.gitignore` or run `tar_git_status_data()` to restore the stash automatically if no `.gitignore` already exists.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status_targets()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's files pace.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
list.files("_targets", all.files = TRUE)
gert::git_init()
tar_git_init()
tar_git_status()
})
}
```

tar_git_status_code *Status of the code repository (Git)*

Description

Show the Git status of the code repository.

Usage

```
tar_git_status_code(code = getwd())
```

Arguments

code	Character of length 1, directory path to the code repository, usually the root of the `targets` project.

Value

If the code repository exists, the return value is the data frame produced by `gert::git_status(repo = code)`. If the code has no Git repository, then the return value is `NULL`.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_data()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's file space.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
list.files("_targets", all.files = TRUE)
gert::git_init()
tar_git_init()
tar_git_status_code()
})
}
```

tar_git_status_data *Status of the data repository (Git)*

Description

Show the Git status of the data repository.

Usage

```
tar_git_status_data(
  store = targets::tar_config_get("store"),
  stash_gitignore = TRUE
)
```

Arguments

`store`	Character of length 1, path to the data store of the pipeline. If `NULL`, the `store` setting is left unchanged in the YAML configuration file (default: `_targets.yaml`). Usually, the data store lives at `_targets`. Set `store` to a custom directory to specify a path other than `_targets/`. The path need not exist before the pipeline begins, and it need not end with "_targets", but it must be writeable. For optimal performance, choose a storage location with fast read/write access. If the argument `NULL`, the setting is not modified. Use `tar_config_unset()` to delete a setting.
`stash_gitignore`	Logical of length 1, whether to temporarily stash the `.gitignore` file of the data store. See the "Stashing .gitignore" section for details.

Value

If the data repository exists, the return value is the data frame produced by `gert::git_status(repo = store)`. If the data store has no Git repository, then the return value is `NULL`.

Stashing .gitignore

The `targets` package writes a `.gitignore` file to new data stores in order to prevent accidental commits to the code Git repository. Unfortunately, for `gittargets`, this automatic `.gitignore` file interferes with proper data versioning. So by default, `gittargets` temporarily stashes it to a hidden file called `.gittargets_gitignore` inside the data store. If your R program crashes while the stash is active, you can simply move it manually back to `.gitignore` or run `tar_git_status_data()` to restore the stash automatically if no `.gitignore` already exists.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_targets()`, `tar_git_status()`

Examples

```
if (Sys.getenv("TAR_EXAMPLES") == "true" && tar_git_ok(verbose = FALSE)) {
targets::tar_dir({ # Containing code does not modify the user's file space.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
list.files("_targets", all.files = TRUE)
gert::git_init()
tar_git_init()
tar_git_status_data()
})
}
```

`tar_git_status_targets` *Status of the targets (Git)*

Description

Show which targets are outdated.

Usage

```
tar_git_status_targets(
  script = targets::tar_config_get("script"),
  store = targets::tar_config_get("store"),
  reporter = targets::tar_config_get("reporter_outdated"),
  envir = parent.frame(),
  callr_function = callr::r,
  callr_arguments = NULL
)
```

Arguments

`script`	Character of length 1, path to the target script file. Defaults to `tar_config_get("script")`, which in turn defaults to `_targets.R`. When you set this argument, the value of `tar_config_get("script")` is temporarily changed for the current function call. See `tar_script()`, `tar_config_get()`, and `tar_config_set()` for details about the target script file and how to set it persistently for a project.
`store`	Character of length 1, path to the `targets` data store. Defaults to `tar_config_get("store")`, which in turn defaults to `_targets/`. When you set this argument, the value of `tar_config_get("store")` is temporarily changed for the current function call. See `tar_config_get()` and `tar_config_set()` for details about how to set the data store path persistently for a project.
`reporter`	Character of length 1, name of the reporter to user. Controls how messages are printed as targets are checked. Choices: • `"silent"`: print nothing.

- `"forecast"`: print running totals of the checked and outdated targets found so far.

`envir`	An environment, where to run the target R script (default: `_targets.R`) if `callr_function` is `NULL`. Ignored if `callr_function` is anything other than `NULL`. `callr_function` should only be `NULL` for debugging and testing purposes, not for serious runs of a pipeline, etc. The `envir` argument of `tar_make()` and related functions always overrides the current value of `tar_option_get("envir")` in the current R session just before running the target script file, so whenever you need to set an alternative `envir`, you should always set it with `tar_option_set()` from within the target script file. In other words, if you call `tar_option_set(envir = envir1)` in an interactive session and then `tar_make(envir = envir2, callr_function = NULL)`, then `envir2` will be used.
`callr_function`	A function from `callr` to start a fresh clean R process to do the work. Set to `NULL` to run in the current session instead of an external process (but restart your R session just before you do in order to clear debris out of the global environment). `callr_function` needs to be `NULL` for interactive debugging, e.g. `tar_option_set(debug = "your_target")`. However, `callr_function` should not be `NULL` for serious reproducible work.
`callr_arguments`	A list of arguments to `callr_function`.

Details

This function has prettier output than `targets::tar_outdated()`, and it mainly serves `tar_git_status()`.

Value

A `tibble` with the names of outdated targets.

See Also

Other git: `tar_git_checkout()`, `tar_git_init()`, `tar_git_log()`, `tar_git_ok()`, `tar_git_snapshot()`, `tar_git_status_code()`, `tar_git_status_data()`, `tar_git_status()`

Examples

```
targets::tar_dir({ # Containing code does not modify the user's file space.
targets::tar_script(tar_target(data, 1))
targets::tar_make()
list.files("_targets", all.files = TRUE)
tar_git_status_targets()
})
```

Index