# Package 'drake'

May 10, 2017

**Title** Data Frames in R for Make

**Version** 3.0.0

**Date** 2017-05-09

**Description** A solution for reproducible code and high-performance computing.

**License** GPL-3

**Depends** R (>= 3.2.0)

**Imports** base64url, codetools, crayon, eply, digest, igraph, magrittr, parallel, plyr, R.utils, storr, stringi, stringr, testthat, tools, utils

**Suggests** abind, knitr, MASS, rmarkdown, tibble

**VignetteBuilder** knitr

**URL** https://github.com/wlandau-lilly/drake

**BugReports** https://github.com/wlandau-lilly/drake/issues

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** William Michael Landau [aut, cre], Eli Lilly and Company [cph]

**Maintainer** William Michael Landau <will.landau@lilly.com>

**Repository** CRAN

**Date/Publication** 2017-05-09 22:12:20 UTC

## R topics documented:

---

`analyses` *Function* `analyses`

---

## Description

Generate a workflow plan data frame to analyze multiple datasets using multiple methods of analysis.

## Usage

```
analyses(plan, datasets)
```

## Arguments

| | |
|---|---|
| `plan` | workflow plan data frame of analysis methods. The commands in the `command` column must have the `..dataset..` wildcard where the datasets go. For example, one command could be `lm(..dataset..)`. Then, the commands in the output will include `lm(your_dataset_1)`, `lm(your_dataset_2)`, etc. |
| `datasets` | workflow plan data frame with instructions to make the datasets. |

### Value

an evaluated workflow plan data frame of analysis instructions

### See Also

summaries, make, plan

---

## as_file *Function* as_file

---

### Description

Converts an ordinary character string into a filename understandable by drake. In other words, as_file(x) just wraps single quotes around x.

### Usage

```
as_file(x)
```

### Arguments

| | |
|---|---|
| x | character string to be turned into a filename understandable by drake (i.e., a string with literal single quotes on both ends). |

### Value

a single-quoted character string: i.e., a filename understandable by drake.

---

## build_graph *Function* build_graph

---

### Description

Make a graph of the dependency structure of your workflow.

### Usage

```
build_graph(plan, targets = drake::possible_targets(plan),
  envir = parent.frame(), verbose = TRUE)
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame, same as for function make(). |
| targets | names of targets to bulid, same as for function make(). |
| envir | environment to import from, same as for function make(). |
| verbose | logical, whether to output messages to the console. |

### Details

This function returns an igraph object representing how the targets in your workflow depend on each other. (`help(package = "igraph")`). To plot the graph, call to `plot.igraph()` on your graph, or just use `plot_graph()` from the start.

---

## built *Function* `built`

---

### Description

List all the built (non-imported) objects in the drake cache.

### Usage

```
built(path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| `path` | Root directory of the drake project, or if `search` is `TRUE`, either the project root or a subdirectory of the project. |
| `search` | logical. If `TRUE`, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

list of imported objects in the cache

### See Also

`cached`, `loadd`, `link{imported}`

---

## cached *Function* `cached`

---

### Description

Check whether targets are in the cache. If no targets are specified with `...` or `list`, then `cached()` lists all the items in the drake cache. Read/load a cached item with `readd()` or `loadd()`.

### Usage

```
cached(..., list = character(0), no_imported_objects = FALSE,
  path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| `...` | objects to load from the cache, as names (unquoted) or character strings (quoted). Similar to `...` in `remove(...)`. |
| `list` | character vector naming objects to be loaded from the cache. Similar to the `list` argument of `remove()`. |
| `no_imported_objects` | logical, applies only when no targets are specified and a list of cached targets is returned. If `no_imported_objects` is `TRUE`, then `cached()` shows built targets (with commands) plus imported files, ignoring imported objects. Otherwise, the full collection of all cached objects will be listed. Since all your functions and all their global variables are imported, the full list of imported objects could get really cumbersome. |
| `path` | Root directory of the drake project, or if `search` is `TRUE`, either the project root or a subdirectory of the project. |
| `search` | logical. If `TRUE`, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

Either a named logical indicating whether the given targets or cached or a character vector listing all cached items, depending on whether any targets are specified

### See Also

`built`, `imported`, `readd`, `loadd`, `plan`, `make`

---

## check *Function* check

---

### Description

Check a workflow plan, etc. for obvious errors such as circular dependencies and missing input files.

### Usage

```
check(plan, targets = drake::possible_targets(plan), envir = parent.frame())
```

### Arguments

| | |
|---|---|
| `plan` | workflow plan data frame, possibly from `plan()`. |
| `targets` | character vector of targets to make |
| `envir` | environment containing user-defined functions |

## Value

invisibly return plan

## See Also

link{plan}, make

---

clean *Function* clean

---

## Description

Cleans up all work done by make().

## Usage

```
clean(..., list = character(0), destroy = FALSE, path = getwd(),
  search = TRUE)
```

## Arguments

| | |
|---|---|
| ... | targets to remove from the cache, as names (unquoted) or character strings (quoted). Similar to ... in remove(...). |
| list | character vector naming targets to be removed from the cache. Similar to the list argument of remove(). |
| destroy | logical, whether to totally remove the drake cache. If destroy is FALSE, only the targets from make() are removed. If TRUE, the whole cache is removed, including session metadata, etc. |
| path | Root directory of the drake project, or if search is TRUE, either the project root or a subdirectory of the project. |
| search | logical. If TRUE, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

## Details

You must be in your project's working directory or a subdirectory of it. clean(search = TRUE) searches upwards in your folder structure for the drake cache and acts on the first one it sees. Use search == FALSE to look within the current working directory only. WARNING: This deletes ALL work done with make(), which includes file targets as well as the entire drake cache. Only use clean() if you're sure you won't lose anything important.

## See Also

prune, make,

## default_parallelism *Function* default_parallelism

### Description

Default parallelism for make(): "parLapply" for Windows machines and "mclapply" for other platforms.

### Usage

```
default_parallelism()
```

### Value

default parallelism option for the current platform

## default_system2_args *Internal function* default_system2_args

### Description

Internal function to configure arguments to system2() to run Makefiles. Not a user-side function.

### Usage

```
default_system2_args(jobs, verbose)
```

### Arguments

| | |
|---|---|
| jobs | number of jobs |
| verbose | logical, whether to be verbose |

### Value

args for system2(command, args)

`evaluate` *Function* `evaluate`

## Description

The commands in workflow plan data frames can have wildcard symbols that can stand for datasets, parameters, function arguments, etc. These wildcards can be evaluated over a set of possible values using `evaluate`.

## Usage

```
evaluate(plan, rules = NULL, wildcard = NULL, values = NULL,
  expand = TRUE)
```

## Arguments

| | |
|---|---|
| `plan` | workflow plan data frame, similar to one produced by `link{plan}` |
| `rules` | Named list with wildcards as names and vectors of replacements as values. This is a way to evaluate multiple wildcards at once. When not `NULL`, `rules` overrules `wildcard` and `values` if not `NULL`. |
| `wildcard` | character scalar denoting a wildcard placeholder |
| `values` | vector of values to replace the wildcard in the drake instructions. Will be treated as a character vector. Must be the same length as `plan$command` if `expand` is `TRUE`. |
| `expand` | If `TRUE`, create a new rows in the workflow plan data frame if multiple values are assigned to a single wildcard. If `FALSE`, each occurence of the wildcard is replaced with the next entry in the `values` vector, and the values are recycled. |

## Details

Specify a single wildcard with the `wildcard` and `values` arguments. In each command, the text in `wildcard` will be replaced by each value in `values` in turn. Specify multiple wildcards ith the `rules` argument, which overrules `wildcard` and `values` if not `NULL`. Here, `rules` should be a list with wildcards as names and vectors of possible values as list elements.

## Value

a workflow plan data frame with the wildcards evaluated

## examples_drake *Function* examples_drake

### Description

List the names of all the drake examples.

### Usage

```
examples_drake()
```

### Value

names of all the drake examples.

### See Also

example_drake, make

## example_drake *Function* example_drake

### Description

Copy a folder of code files for a drake example to the current working directory. To see the names of all the examples, run examples_drake.

### Usage

```
example_drake(example = drake::examples_drake(), destination = getwd())
```

### Arguments

| | |
|---|---|
| example | name of the example. To see all the available example names, run examples_drake. |
| destination | character scalar, file path, where to write the folder containing the code files for the example. |

### See Also

examples_drake, make

## expand *Function* expand

### Description

Expands a workflow plan data frame by duplicating rows. This generates multiple replicates of targets with the same commands.

### Usage

```
expand(plan, values = NULL)
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame |
| values | values to expand over. These will be appended to the names of the new targets. |

### Value

an expanded workflow plan data frame

## find_cache *Function* find_cache

### Description

Return the file path of the nearest drake cache (searching upwards for directories containing a drake cache).

### Usage

```
find_cache(path = getwd())
```

### Arguments

| | |
|---|---|
| path | starting path for search back for the cache. Should be a subdirectory of the drake project. |

### Value

File path of the nearest drake cache or NULL if no cache is found.

### See Also

plan, make,

## find_project *Function* find_project

### Description

Return the file path of the nearest drake project (searching upwards for directories containing a drake cache).

### Usage

```
find_project(path = getwd())
```

### Arguments

| | |
|---|---|
| path | starting path for search back for the project. Should be a subdirectory of the drake project. |

### Value

File path of the nearest drake project or NULL if no drake project is found.

### See Also

plan, make

## gather *Function* gather

### Description

Create a new workflow plan data frame with a single new target. This new target is a list, vector, or other aggregate of a collection of existing targets in another workflow plan data frame.

### Usage

```
gather(plan = NULL, target = "target", gather = "list")
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame of prespecified targets |
| target | name of the new aggregated target |
| gather | function used to gather the targets. Should be one of list(...), c(...), rbind(...), or similar. |

### Value

workflow plan data frame for aggregating prespecified targets

## imported *Function* imported

### Description

List all the imported objects in the drake cache

### Usage

```
imported(files_only = FALSE, path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| files_only | logical, whether to show imported files only and ignore imported objects. Since all your functions and all their global variables are imported, the full list of imported objects could get really cumbersome. |
| path | Root directory of the drake project, or if search is TRUE, either the project root or a subdirectory of the project. |
| search | logical. If TRUE, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

character vector naming the imported objects in the cache

### See Also

cached, loadd, built

## loadd *Function* loadd

### Description

Load object(s) from the drake cache into the current workspace (or envir if given). Defaults to loading the whole cache if arguments ... and list are not set (or all the imported objects if in addition imported_only is TRUE).

### Usage

```
loadd(..., list = character(0), imported_only = FALSE, path = getwd(),
  search = TRUE, envir = parent.frame())
```

### Arguments

| | |
|---|---|
| ... | targets to load from the cache, as names (unquoted) or character strings (quoted). Similar to ... in remove(...). |
| list | character vector naming targets to be loaded from the cache. Similar to the list argument of remove(). |
| imported_only | logical, whether only imported objects should be loaded. |
| path | Root directory of the drake project, or if search is TRUE, either the project root or a subdirectory of the project. |
| search | logical. If TRUE, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |
| envir | environment to load objects into. Defaults to the calling environment (current workspace). |

### See Also

cached, built, imported, plan, make,

---

## make *Function* make

---

### Description

Run your project (build the targets).

### Usage

```
make(plan, targets = drake::possible_targets(plan), envir = parent.frame(),
  verbose = TRUE, parallelism = drake::default_parallelism(), jobs = 1,
  packages = (.packages()), prework = character(0),
  prepend = character(0), command = "make",
  args = drake::default_system2_args(jobs = jobs, verbose = verbose))
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame. A workflow plan data frame is a data frame with a target column and a command column. Targets are the objects and files that drake generates, and commands are the pieces of R code that produce them. Use the function plan() to generate workflow plan data frames easily, and see functions analyses(), summaries(), evaluate(), expand(), and gather() for easy ways to generate large workflow plan data frames. |
| targets | character string, names of targets to build. Dependencies are built too. |
| envir | environment to use. Defaults to the current workspace, so you should not need to worry about this most of the time. A deep copy of envir is made, so you don't need to worry about your workspace being modified by make. The deep copy inherits from the global environment. Wherever necessary, objects and functions are imported from envir and the global environment and then reproducibly tracked as dependencies. |

verbose — logical, whether to print progress to the console. Skipped objects are not printed.

parallelism — character, type of parallelism to use. See parallelism_choices() for the choices for this argument, and run ?parallelism_choices for an explanation of each. To use parallelism at all, be sure that jobs >= 2. If parallelism is "mclapply", drake will use parallel::mclapply() to distribute targets across parallel processes wherever possible. (This is not possible on Windows.) Setting parallelism to "parLapply" is similar, except that it uses parallel::parLapply(). It works on Windows, but it requires more overhead (except if jobs == 1, in which case no "cluster" is created). If "Makefile", drake will write and execute a Makefile to distribute targets across separate R sessions. The vignettes (vignette(package = "drake")) show how to turn those R sessions into separate jobs on a cluster. Read the vignettes to learn how to take advantage of multiple nodes on a supercomputer. WARNING: the Makefile is NOT standalone. Do not run outside of make(). For parallelism == "Makefile", Windows users will need to download and install Rtools.

jobs — number of parallel processes or jobs to run. Windows users should not set jobs > 1 if parallelism is "mclapply" because mclapply() is based on forking. Windows users who use parallelism == "Makefile" will need to download and install Rtools.

packages — character vector packages to load, in the order they should be loaded. Defaults to (.packages()), so you shouldn't usually need to set this manually. Just call library() to load your packages before make(). However, sometimes packages need to be strictly forced to load in a certian order, especially if parallelism is "Makefile". To do this, do not use library() or require() or loadNamespace() or attachNamespace() to load any libraries beforehand. Just list your packages in the packages argument in the order you want them to be loaded. If parallelism is "mclapply", the necessary packages are loaded once before any targets are built. If parallelism is "Makefile", the necessary packages are loaded once on initialization and then once again for each target right before that target is built.

prework — character vector of lines of code to run before build time. This code can be used to load packages, set options, etc., although the packages in the packages argument are loaded before any prework is done. If parallelism is "mclapply", the prework is run once before any targets are built. If parallelism is "Makefile", the prework is run once on initialization and then once again for each target right before that target is built.

prepend — lines to prepend to the Makefile if parallelism is "Makefile". See the vignettes (vignette(package = "drake")) to learn how to use prepend to take advantage of multiple nodes of a supercomputer.

command — character scalar, command to call the Makefile generated for distributed computing. Only applies when parallelism is "Makefile". Defaults to the usual "make", but it could also be "lsmake" on supporting systems, for example. command and args are executed via system2(command, args) to run the Makefile. If args has something like "--jobs=2", or if jobs >= 2 and args is left alone, targets will be distributed over independent parallel R sessions wherever possible.

`args` command line arguments to call the Makefile for distributed computing. For advanced users only. If set, `jobs` and `verbose` are overwritten as they apply to the Makefile. `command` and `args` are executed via `system2(command, args)` to run the Makefile. If `args` has something like `"--jobs=2"`, or if `jobs >= 2` and `args` is left alone, targets will be distributed over independent parallel R sessions wherever possible.

---

## mk *Function* mk

---

### Description

Internal drake function to be called inside Makefiles only. Makes a single target. Users, do not invoke directly.

### Usage

```
mk(target)
```

### Arguments

`target` name of target to make

---

## parallelism_choices *Function* parallelism_choices

---

### Description

List the types of supported parallel computing.

### Usage

```
parallelism_choices()
```

### Details

Run `make(..., parallelism = x, jobs = n)` for any of the following values of `x` to distribute targets over parallel units of execution.

**"parLapply"** launches multiple processes in a single R session using `parallel::parLapply()`. This is single-node, (potentially) multicore computing. It requires more overhead than the `"mclapply"` option, but it works on Windows. If `jobs` is `1` in `make()`, then no "cluster" is created and no parallelism is used.

**"mclapply"** uses multiple processes in a single R session. This is single-node, (potentially) multicore computing. Does not work on Windows for `jobs > 1` because `mclapply()` is based on forking.

**"Makefile"** uses multiple R sessions by creating and running a Makefile. The Makefile is NOT standalone. DO NOT run outside of `make()` or `make()`. Windows users will need to download and intall Rtools. As explained in the vignettes, you can use the `prepend` to `make()` or `make()` to distribute targets over multiple nodes of a supercomputer. Use this approach for true distributed computing.

## Value

Character vector listing the types of parallel computing supported.

---

`plan` *Function* `plan`

---

## Description

Turns a named collection of command/target pairs into a workflow plan data frame for `make` and `check`.

## Usage

```
plan(..., list = character(0), file_targets = FALSE,
  strings_in_dots = c("filenames", "literals"))
```

## Arguments

| | |
|---|---|
| `...` | commands named by the targets they generate. Recall that drake uses single quotes to denote external files and double-quoted strings as ordinary strings. Use the `strings_in_dots` argument to control the quoting in `....` |
| `list` | character vector of commands named by the targets they generate. |
| `file_targets` | logical. If `TRUE`, targets are single-quoted to tell drake that these are external files that should be expected as output in the next call to `make()`. |
| `strings_in_dots` | character scalar. If `"filenames"`, all character strings in `...` will be treated as names of file dependencies (single-quoted). If `"literals"`, all character strings in `...` will be treated as ordinary strings, not dependencies of any sort (double-quoted). Because of R's automatic parsing/deparsing behavior, strings in `...` cannot simply be left alone. |

## Details

A workflow plan data frame is a data frame with a `target` column and a `command` column. Targets are the objects and files that drake generates, and commands are the pieces of R code that produce them.

For file inputs and targets, drake uses single quotes. Double quotes are reserved for ordinary strings. The distinction is important because drake thinks about how files, objects, targets, etc. depend on each other. Quotes in the `list` argument are left alone, but R messes with quotes when it parses the freeform arguments in `...`, so use the `strings_in_dots` argument to control the quoting in `....`

### Value

data frame of targets and command

### See Also

link{check}, make,

---

## plot_graph *Function* plot_graph

---

### Description

Plot the dependency structure of your workflow

### Usage

```
plot_graph(plan, targets = drake::possible_targets(plan),
  envir = parent.frame(), verbose = TRUE)
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame, same as for function make(). |
| targets | names of targets to bulid, same as for function make(). |
| envir | environment to import from, same as for function make(). |
| verbose | logical, whether to output messages to the console. |

---

## possible_targets *Function* possible_targets

---

### Description

internal function, returns the list of possible targets that you can select with the targets argument to make().

### Usage

```
possible_targets(plan)
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame |

### Value

character vector of possible targets

## See Also

make

---

prune *Deprecated function* prune

---

## Description

Use clean() instead

## Usage

```
prune(plan)
```

## Arguments

| | |
|---|---|
| plan | workflow plan data frame, as generated by plan. |

## See Also

clean, make

---

readd *Function* readd

---

## Description

Read a drake target object from the cache. Does not delete the item from the cache.

## Usage

```
readd(target, character_only = FALSE, path = getwd(), search = TRUE,
  cache = NULL)
```

## Arguments

| | |
|---|---|
| target | If character_only is TRUE, target is a character string naming the object to read. Otherwise, target is an unquoted symbol with the name of the object. Note: target could be the name of an imported object. |
| character_only | logical, whether name should be treated as a character or a symbol (just like character.only in library()). |
| path | Root directory of the drake project, or if search is TRUE, either the project root or a subdirectory of the project. |
| search | logical. If TRUE, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |
| cache | a storr cache. Mainly for internal use. |

## Value

drake target item from the cache

## See Also

loadd, cached, built, link{imported}, plan, make

---

read_config *Function* read_config

---

## Description

Read all the configuration parameters from your last attempted call to make(). These include the workflow plan

## Usage

```
read_config(path = getwd(), search = TRUE)
```

## Arguments

| | |
|---|---|
| path | Root directory of the drake project, or if search is TRUE, either the project root or a subdirectory of the project. |
| search | logical. If TRUE, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

## Value

a named list of configuration items

## See Also

make

## read_graph *Function* read_graph

### Description

Read the dependency graph of your targets from your last attempted call to `make()`.

### Usage

```
read_graph(plot = TRUE, path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| `plot` | logical, whether to plot the graph or simply return the graph as an igraph object. |
| `path` | Root directory of the drake project, or if `search` is `TRUE`, either the project root or a subdirectory of the project. |
| `search` | logical. If `TRUE`, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

either a plot or an igraph object, depending on `plot`

### See Also

`read_config`

## read_plan *Function* read_plan

### Description

Read the workflow plan from your last attempted call to `make()`.

### Usage

```
read_plan(path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| `path` | Root directory of the drake project, or if `search` is `TRUE`, either the project root or a subdirectory of the project. |
| `search` | logical. If `TRUE`, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

a workflow plan data frame

### See Also

read_config

---

## session *Function* session

---

### Description

Load the sessionInfo() of the last call to make().

### Usage

```
session(path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| path | Root directory of the drake project, or if search is TRUE, either the project root or a subdirectory of the project. |
| search | If TRUE, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

sessionInfo() of the last call to make()

### See Also

built, imported, readd, plan, make

---

## status *Function* status

---

### Description

Get the build status (overall or individual targets) of the last call to make() or make(). Objects that drake imported, built, or attempted to build are listed as "finished" or "in progress". Skipped objects are not listed.

### Usage

```
status(..., list = character(0), imported_files_only = FALSE,
  path = getwd(), search = TRUE)
```

### Arguments

| | |
|---|---|
| `...` | objects to load from the cache, as names (unquoted) or character strings (quoted). Similar to `...` in `remove(...)`. |
| `list` | character vector naming objects to be loaded from the cache. Similar to the `list` argument of `remove()`. |
| `imported_files_only` | logical, applies only when no targets are specified and the statuses of cached targets are returned. If `imported_files_only` logical, whether to ignore imported objects that are not files. If `TRUE`, all targets (with commands in the workflow plan data frame) and imported files will be listed. Otherwise, everything will be listed. |
| `path` | Root directory of the drake project, or if `search` is `TRUE`, either the project root or a subdirectory of the project. |
| `search` | If `TRUE`, search parent directories to find the nearest drake cache. Otherwise, look in the current working directory only. |

### Value

Either the build status of each target given (from the last call to `make()` or `make()`), or if no targets are specified, a data frame containing the build status of the last session. In the latter case, only finished targets are listed.

Either a named logical indicating whether the given targets or cached or a character vector listing all cached items, depending on whether any targets are specified

### See Also

`session`, `built`, `imported`, `readd`, `plan`, `make`

---

## `summaries` *Function* `summaries`

---

### Description

Generate a worklfow plan data frame for summarizing multiple analyses of multiple datasets multiple ways.

### Usage

```
summaries(plan, analyses, datasets, gather = rep("list", nrow(plan)))
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame with commands for the summaries. Use the ..analysis.. and ..dataset.. wildcards just like the ..dataset.. wildcard in analyses(). |
| analyses | workflow plan data frame of analysis instructions |
| datasets | workflow plan data frame with instructions to make or import the datasets. |
| gather | Character vector, names of functions to gather the summaries. If not NULL, the length must be the number of rows in the plan. See the gather() function for more. |

### Value

an evaluated workflow plan data frame of instructions for computing summaries of analyses and datasets. analyses of multiple datasets in multiple ways.

### See Also

analyses, make, plan

---

## tracked *Function* tracked

---

### Description

Print out which objects, functions, files, targets, etc. are reproducibly tracked.

### Usage

```
tracked(plan, targets = drake::possible_targets(plan),
  envir = parent.frame())
```

### Arguments

| | |
|---|---|
| plan | workflow plan data frame, same as for function make(). |
| targets | names of targets to bulid, same as for function make(). |
| envir | environment to import from, same as for function make(). |

# Index