

Package: ggtargets (via r-universe)

October 16, 2026

Title Metro-Style Diagrams of 'targets' Pipeline Dependency Graphs

Version 0.0.0.9001

Date 2026-10-16

Maintainer Roy Francis <roy.m.francis@outlook.com>

Description Create metro-style diagrams of targets pipelines
dependency graph and other directed acyclic graphs using
ggplot2.

URL <https://github.com/royfrancis/ggtargets>

<https://royfrancis.github.io/ggtargets>

<https://royfrancis.r-universe.dev/ggtargets>

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Imports ggplot2 (>= 4.0.0), grDevices, grid

Suggests covr, igraph, knitr, png, quarto, svglite, targets (>= 1.6.0), testthat (>= 3.2.0), tidyselect, vdiff, withr

BugReports <https://github.com/royfrancis/ggtargets/issues>

VignetteBuilder quarto

Config/testthat/edition 3

Config/Needs/website viridisLite, RColorBrewer, randomcoloR, patchwork

RoxygenNote 7.3.3

Repository <https://royfrancis.r-universe.dev>

Date/Publication 2026-10-16 15:24:37 UTC

RemoteUrl <https://github.com/royfrancis/ggtargets>

RemoteRef main

RemoteSha 02011a56125a41c6c78c287b9a39661b521177a6

Contents

as_dag	2
dag_collapse	3
dag_layers	5
dag_layout	7
dag_size	8
dag_theme	8
ggplot_dag	9
ggsave_dag	11
metro_palette	12
style_metro	12
tar_dag	14
tar_ggplot	15
targets_node_shapes	18
targets_status_colors	19
Index	20

as_dag	<i>Build a dag from an edge list or other graph representation</i>
--------	---

Description

as_dag() is the entry point for any DAG that isn’t read from a targets pipeline (see tar_dag()). It normalizes the input into a dag: a classed list with nodes and edges data frames, keeping insertion order throughout, the way networkx does.

Usage

```
as_dag(x, ...)  
  
## S3 method for class 'data.frame'  
as_dag(x, nodes = NULL, ...)  
  
## S3 method for class 'list'  
as_dag(x, ...)  
  
## Default S3 method:  
as_dag(x, ...)
```

Arguments

- x An edge-list data frame with from and to columns, or a list with vertices and edges elements (the shape targets::tar_network() returns; see tar_dag()). Any other columns of the edge-list data frame are kept as edge attributes; a numeric one can set the edges’ width (edge_width, see ggplot_dag()).
- ... Passed on to methods.

nodes Optional data frame of nodes, with at least a name column, listed before any additional nodes implied only by `x`'s endpoints. Its other columns are node attributes, but not named `row`, `col`, `x`, `y` or `color_index`: the layout uses those names for its own columns (see [dag_layout\(\)](#)), so these names are reserved, and using one is an error.

Details

`as_dag.list()` deliberately narrows the return value described above. It is the [tar_dag\(\)](#) path, so it keeps only the vertices columns a metro plot can use - `type`, `status`, `description`, `seconds`, `bytes` and `branches` - and no extra edges columns. To color by other node attributes, pass an edge-list data frame and a nodes table to `as_dag.data.frame()` instead.

Node names are identities, and the package matches edge colors on them downstream, so every method rejects a name that is NA, empty or contains `"|"` or a carriage return.

Value

A dag: a classed list with nodes (columns `name`, `label`, plus any extra attributes, which `node_color`, `node_shape` and, for a numeric one, `node_size` can name) and edges (columns `from`, `to`, plus any extra columns of `x`, which only `edge_width` uses for drawing) data frames, in insertion order.

Examples

```
edges <- data.frame(from = c("a", "a", "b"), to = c("b", "c", "c"))
dag <- as_dag(edges)
print(dag)
ggplot_dag(dag)

# a node table, with extra attributes for coloring
nodes <- data.frame(name = c("a", "b", "c"), type = c("x", "y", "x"))
dag2 <- as_dag(edges, nodes = nodes)
ggplot_dag(dag2, node_color = "type")
```

dag_collapse

Collapse a DAG by a node column

Description

Collapses a dag: the nodes that share a value of the `by` column are merged into one node, named after that value, and the edges are projected onto these new nodes. A pipeline of fifty targets in four stages becomes a graph of four nodes, drawn by the same functions ([ggplot_dag\(\)](#), [tar_ggplot\(\)](#)) as any other dag.

Usage

```
dag_collapse(dag, by, label_n = TRUE)
```

Arguments

<code>dag</code>	A dag, from <code>as_dag()</code> or <code>tar_dag()</code> .
<code>by</code>	The name of a node column (a single string). Its values are used as text.
<code>label_n</code>	If TRUE (the default), a merged node's label ends with its member count, "model (7)"; otherwise it is just the value.

Details

Any node column works as `by`, so there is nothing special about its name: a column of your own (passed to `as_dag()` as `nodes`), or one that `tar_dag()` returns - `description`, `type` or `status`. `targets` has no tag or category field of its own, and `description` is its one free-form text, so a pipeline's stages go there: see "Collapsing a pipeline by stage" in `tar_ggplot()`. For any other metadata, add a column to `dag$nodes` yourself first (not named `row`, `col`, `x`, `y` or `color_index`).

A node with no value in `by` (NA or an empty string) stays on its own, and a value that is also the name of such a node is an error, since the two would be merged.

- **Nodes.** `name` is the value of `by`, and `label` is that value with the member count, "model (7)". A node with no `by` value is not merged: it stays as it is, in its own place. The order is the order of first appearance, so each merged node sits where its first member was. `n` counts the members (replacing any `n` column you already have, so `n` itself is not a useful `by`).
- **Other node columns.** `seconds`, `bytes` and `branches` are summed (NA if every member is NA). Any other column keeps its value when all the members share it, and is NA otherwise; in particular the `by` column itself keeps its value (even if it is one of the columns that are otherwise summed), so `node_color = by` colors the collapsed graph.
- **Edges.** An edge between two members of one group disappears. The others are drawn between the groups, once per pair of groups, with an `n` column that counts the edges it stands for. Any other edge column is dropped.

Merging nodes can create a cycle, which a DAG can't have: with `a -> b -> c`, `a` and `c` in group `X` and `b` in group `Y`, the collapsed graph has `X -> Y -> X`. That is an error, naming the groups and the edges behind it. Group by a different column, or move the node that closes the loop.

Value

A dag, with the columns described above: `nodes` `name`, `label`, `n` and the rolled-up columns, and edges `from`, `to` and `n`.

Examples

```
nodes <- data.frame(
  name = c("raw", "clean", "fit", "eval", "report"),
  stage = c("data", "data", "model", "model", "report")
)
edges <- data.frame(
  from = c("raw", "clean", "fit", "eval"),
  to = c("clean", "fit", "eval", "report")
)
dag <- as_dag(edges, nodes = nodes)
small <- dag_collapse(dag, "stage")
```

```

small$nodes
small$edges
ggplot_dag(small, node_color = "stage")

# a grouping that closes a loop is an error
bad <- as_dag(
  data.frame(from = c("a", "b"), to = c("b", "c")),
  nodes = data.frame(name = c("a", "b", "c"), group = c("X", "Y", "X"))
)
try(dag_collapse(bad, "group"))

# the targets' own `description` as the key
targets::tar_dir({
  targets::tar_script(list(
    targets::tar_target(x, 1, description = "input"),
    targets::tar_target(y, x + 1, description = "input"),
    targets::tar_target(z, y * 2, description = "output")
  ))
  ggplot_dag(dag_collapse(tar_dag(), "description"))
})

```

dag_layers

Draw a DAG layout as ggplot2 layers

Description

Returns a list of ggplot2 layers in draw order, ready to add to a ggplot() call (or combined with [dag_theme\(\)](#) by [ggplot_dag\(\)](#)).

Usage

```

dag_layers(
  layout,
  node_color = NULL,
  edge_color = "source",
  palette = metro_palette,
  node_shape = NULL,
  shapes = targets_node_shapes(),
  node_size = NULL,
  edge_width = NULL,
  zoom = 1
)

```

Arguments

layout A dag_layout, as returned by [dag_layout\(\)](#).

node_color, palette, edge_color	Color resolution arguments: see the pkgdown site's Colors article for the full set of rules and examples. palette must supply at least one color per node (or per distinct value of a node_color column); colors are never recycled, and the default <code>metro_palette()</code> never repeats one. palette = NULL is the same as the default, <code>metro_palette()</code> .
node_shape	The shape of a node: NULL (the default), a node column name, or a single shape name. A shape is "circle", "square", "diamond", "triangle" or "triangle_down". NULL follows the nodes' type column if they have one (as a pipeline from <code>tar_dag()</code> does), and otherwise draws circles; a type value that shapes doesn't name is a circle. A column name maps that column to shapes and warns about values shapes doesn't name; a shape name draws every node as that shape. A legend names the shapes whenever more than one is drawn; with node_color set to the same column, it is merged with the color legend. Remove it with <code>+ ggplot2::guides(shape = "none")</code> .
shapes	Which shape each value of the node_shape column gets: a named character vector (value -> shape), or an unnamed one of shapes assigned to the distinct values in order, which must supply at least one per value. Defaults to <code>targets_node_shapes()</code> .
node_size	The size of a node: NULL (the default, every node has the radius node_radius), or the name of a numeric node column, such as the n that <code>dag_collapse()</code> adds (or seconds or bytes of a pipeline from <code>tar_dag()</code>). The radius follows the square root of the value's position between the smallest and the largest value, over the node_size_range of <code>style_metro()</code> , so that it is about the node's area that grows in step with the value. A node with an NA value has the radius node_radius. There is no legend for the size: for a collapsed graph, the labels carry the count. The largest radius a circle may have keeps it inside the grid, but a triangle or diamond reaches further (see node_radius of <code>style_metro()</code>), so with a node_shape of those, the biggest ones in the first or last row, or the left column, may be cut off: raise scale. Two of the biggest nodes in neighboring rows leave almost no room for the line between them: see node_size_range of <code>style_metro()</code> .
edge_width	The width of an edge: NULL (the default, every edge is edge_stroke_width wide), or the name of a numeric column of the dag's edges, such as the n of <code>dag_collapse()</code> . The width grows linearly with the value, over the edge_width_range of <code>style_metro()</code> . An edge with an NA value is edge_stroke_width. In this metro layout the edges from one node share a line, and so do those entering one node from the same side, so a shared line is as wide as the widest edge it carries. Each edge is then drawn whole (as for any edge_color but "source"), the widest first, so that the narrower ones stay visible on top; and each bridge is wide enough for the widest edge drawn over it.
zoom	Multiplier for node size, line widths and text size, to match a panel sized at zoom px per data unit (see <code>ggplot_dag()</code>).

Value

A list of ggplot2 layers.

Examples

```
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
lay <- dag_layout(dag, style_metro())
ggplot2::ggplot() + dag_layers(lay) + dag_theme()
```

dag_layout	<i>Lay out a DAG for drawing</i>
------------	----------------------------------

Description

Turns an `as_dag()` dag into a `dag_layout`, the plain data that `dag_layers()` and `ggplot_dag()` draw from. Coordinates are in dagviz's SVG pixel space, with y growing downward (`dag_theme()` reverses the y scale to match).

Usage

```
dag_layout(dag, style = style_metro())
```

Arguments

dag	A dag, as returned by <code>as_dag()</code> or <code>tar_dag()</code> .
style	A <code>style_metro()</code> object.

Value

A `dag_layout`: a classed list of

- nodes: one row per node, in row order - name, label, row, col, x, y, color_index, plus the dag's extra node columns. Those can't be named row, col, x, y or color_index: the layout needs these, so such a column is an error.
- segments: straight lines, layer ("leader", "vline", "bridge" or "hline"), x, y, xend, yend, from, to and order (dagviz's draw order, shared with arcs).
- arcs: one point path per arc (id), with x, y, from, to and order.
- labels: name, label, x, y.
- edge_paths: one full point path per graph edge - from, to, piece ("vline", "arc" or "hline"), x, y, and order (point order along the path) - used for edge_color other than "source", for any edge_width, and for highlighting paths.
- edges: the dag's edges as they are - from, to, plus any extra edge columns, such as the n of `dag_collapse()`, which edge_width of `dag_layers()` can name. (They are kept apart from edge_paths, so that no column of yours can overwrite a coordinate there.)
- viewbox: a list of left, top, bottom, right, width and height.
- style: the `style_metro()` object.

Examples

```
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
lay <- dag_layout(dag, style_metro())
print(lay)
lay$nodes
```

dag_size	<i>The size of a DAG plot</i>
----------	-------------------------------

Description

For a plot drawn with a fixed panel size (the `ggplot_dag()` default), returns the full plot's size, including any legend or title added with `ggplot2::labs()`; a legend taller than the graph makes the plot taller, so it isn't cut off (with a few percent to spare, as devices measure text slightly differently). Otherwise, returns the view box size at 1 px per data unit (96 dpi), read from the `coord_fixed()` limits that `dag_layers()` sets. Useful for sizing figures, e.g. knitr's `fig.width/fig.height`.

Usage

```
dag_size(plot)
```

Arguments

plot A plot from `ggplot_dag()` or `tar_ggplot()`.

Value

A named numeric vector, `c(width =, height =)`, in inches.

Examples

```
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
dag_size(ggplot_dag(dag))
dag_size(ggplot_dag(dag, node_color = "name")) # wider: includes the legend
```

dag_theme	<i>The theme and scales for a DAG plot</i>
-----------	--

Description

`theme_void()` with zero plot margins and a white background, plus `scale_y_reverse()`: layout coordinates have y growing downward, as in dagviz's SVG. The theme is an ordinary ggplot2 theme, so any of its elements can be overridden by adding `theme()` settings afterwards.

Usage

```
dag_theme()
```

Details

A legend gets a small margin on its right (`legend.box.margin`, 5.5 pt), so that it doesn't touch the edge of the image. A plot without a legend has no such margin.

Value

A list of ggplot2 theme/scale components.

Examples

```
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
lay <- dag_layout(dag, style_metro())
ggplot2::ggplot() + dag_layers(lay) + dag_theme()
```

ggplot_dag

*Plot a DAG or a DAG layout***Description**

`ggplot_dag()` is `ggplot() + dag_layers(lay, ...) + dag_theme()`. Given a dag instead of a `dag_layout`, it first lays it out with `dag_layout()`, in the style you give it (`style_metro()` by default).

Usage

```
ggplot_dag(
  x,
  node_color = NULL,
  edge_color = "source",
  palette = metro_palette,
  node_shape = NULL,
  shapes = targets_node_shapes(),
  node_size = NULL,
  edge_width = NULL,
  style = style_metro(),
  fixed_size = TRUE,
  zoom = 1
)
```

Arguments

<code>x</code>	A <code>dag_layout</code> (from <code>dag_layout()</code>) or a dag (from <code>as_dag()</code> or <code>tar_dag()</code>).
<code>node_color</code> , <code>palette</code> , <code>edge_color</code> , <code>node_shape</code> , <code>shapes</code> , <code>node_size</code> , <code>edge_width</code>	Passed to <code>dag_layers()</code> .
<code>style</code>	A <code>style_metro()</code> object, used to lay out <code>x</code> when it is a dag (ignored for a <code>dag_layout</code> , which carries its own).

<code>fixed_size</code>	If TRUE (the default), fix the panel's absolute size so element sizes are constant across graphs; if FALSE, stretch it to the device.
<code>zoom</code>	Uniform size multiplier for nodes, lines and text (with <code>fixed_size = TRUE</code> only; ignored with a warning otherwise).

Details

By default (`fixed_size = TRUE`) the panel gets an absolute size of `zoom` px (at 96 dpi) per data unit, via `theme(panel.widths, panel.heights)`. Nodes, lines and text then keep the same size whatever the pipeline's length: a bigger graph gives a bigger plot, not smaller nodes. Use `dag_size()` (or `ggsave_dag()`) to get or save the plot at that size. On a device smaller than the plot, such as a small RStudio Plots pane, a large graph is clipped: lower zoom, or save it with `ggsave_dag()`.

With `fixed_size = FALSE` the panel stretches to fill the device, as any ggplot does. Nodes, line widths and text are set in absolute units, so they keep their size while the spacing between nodes stretches, and their proportions to it vary with the graph's size.

Adding a *complete* theme afterwards (e.g. `+ ggplot2::theme_void()`) replaces the panel size and so turns fixed sizing off. Add individual `theme()` settings instead.

Value

A ggplot object.

Examples

```
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
ggplot_dag(dag)
ggplot_dag(dag, edge_color = "target")
ggplot_dag(dag, style = style_metro(node_radius = 9, scale = 12))
ggplot_dag(dag, zoom = 0.5)

# nodes shaped (and colored) by a column
nodes <- data.frame(name = c("a", "b", "c"), role = c("in", "mid", "out"))
ggplot_dag(
  as_dag(data.frame(from = c("a", "b"), to = c("b", "c")), nodes = nodes),
  node_color = "role",
  node_shape = "role",
  shapes = c("diamond", "circle", "triangle")
)
dag_size(ggplot_dag(dag))

# a collapsed graph, with node size and edge width showing the counts
nodes <- data.frame(
  name = c("raw", "clean", "tidy", "fit", "eval", "report"),
  stage = c("data", "data", "data", "model", "model", "report")
)
edges <- data.frame(
  from = c("raw", "clean", "tidy", "tidy", "fit", "eval"),
  to = c("clean", "tidy", "fit", "eval", "eval", "report")
)
small <- dag_collapse(as_dag(edges, nodes = nodes), "stage")
ggplot_dag(small, node_color = "stage", node_size = "n", edge_width = "n")
```

ggsave_dag	<i>Save a DAG plot at its true pixel size</i>
------------	---

Description

Sizes the output with `dag_size()`, so that one data unit equals one SVG pixel (times `ggplot_dag(zoom =)` at scale 1, at 96 dpi).

Usage

```
ggsave_dag(filename, plot = ggplot2::last_plot(), scale = 1, ...)
```

Arguments

filename	File to write to, passed to <code>ggplot2::ggsave()</code> .
plot	A plot from <code>ggplot_dag()</code> (or any ggplot with <code>coord_fixed()</code> limits set the same way, e.g. via manual assembly with <code>dag_layers()</code>).
scale	Resolution multiplier (applied to dpi, not physical size).
...	Passed on to <code>ggplot2::ggsave()</code> .

Details

scale raises resolution ($\text{dpi} = 96 * \text{scale}$) rather than the physical width/height: line widths and text are set in absolute units (points), so scaling the canvas size at a fixed dpi would make them relatively thinner/smaller against the (now larger) node spacing. Scaling dpi instead keeps every element's proportions exactly as dagviz draws them, just at higher resolution.

For a fixed-size plot (the `ggplot_dag()` default) the output also makes room for any legend or title. For a plot with `fixed_size = FALSE` (or manual assembly), the size is the view box's alone, so a legend or title shrinks the panel to fit.

Value

The result of `ggplot2::ggsave()`, invisibly.

Examples

```
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
p <- ggplot_dag(dag)
ggsave_dag(tempfile(fileext = ".png"), p)
```

metro_palette

The default dagviz metro color palette

Description

n distinct colors, never recycled. The first 18 are dagviz.style.colors.palette()'s colors (6 hues at 3 lightness levels, in dagviz's order, minus the 3 duplicates that its 7th hue step produces by wrapping around to the 1st). Further colors are generated deterministically in HCL space, each kept only if it is perceptibly different from all earlier ones. metro_palette(k) is always the first k colors of metro_palette(n) for k <= n. Colors get harder to tell apart as n grows large (beyond roughly 100).

Usage

```
metro_palette(n)
```

Arguments

n Number of colors to generate.

Value

A character vector of n distinct hex colors.

Examples

```
metro_palette(5)

# as a row of color swatches
swatches <- data.frame(x = 1:24, color = metro_palette(24))
ggplot2::ggplot(swatches, ggplot2::aes(x, 1, fill = I(color))) +
  ggplot2::geom_tile() +
  ggplot2::theme_void()
```

style_metro

The metro style

Description

Reproduces dagviz's StyleConfig defaults (style/metro.py) as a dag_style_metro object: round nodes (the default shape; see node_shape of [dag_layers\(\)](#)), thick colored edges with rounded bends, and labels to the right.

Usage

```

style_metro(
  scale = 10,
  node_radius = 6,
  node_fill = NULL,
  node_stroke = "white",
  node_stroke_width = 2,
  edge_stroke_width = 2,
  label_font_family = "sans-serif",
  label_font_size = "inherit",
  label_font_color = "currentColor",
  label_arrow_stroke = "lightgrey",
  label_arrow_dash_array = "2",
  arc_radius = 15,
  bridge_color = "white",
  minimal_width = NULL,
  node_size_range = NULL,
  edge_width_range = NULL
)

```

Arguments

scale	Half the distance, in pixels, between neighboring rows and between neighboring lanes (columns): both are $2 * \text{scale}$ apart.
node_radius	The radius of a node, in pixels: of a circle, and the size that other shapes (see <code>node_shape</code> of <code>dag_layers()</code>) are drawn to match. A triangle or diamond reaches further from its center than a circle, up to about 1.6 times <code>node_radius</code> , and the plot ends only <code>scale</code> pixels from the nodes' centers, so with a <code>node_radius</code> of about 7 or more (at the default scale) a triangle in the first or last row may be cut off: raise <code>scale</code> too.
node_fill	Optional single fill color for every node, used only when <code>node_color</code> (a separate argument to the plotting functions) is left at its default. Unlike <code>node_color</code> , it never affects edge colors (matching dagviz: <code>node_fill</code> recolors <code>place_node()</code> only, never <code>_place_edge()</code>). <code>NULL</code> (the default) uses the resolved palette instead.
node_stroke, node_stroke_width	The color and width of a node's border.
edge_stroke_width	The line width of an edge.
label_font_family, label_font_size, label_font_color	The font family, size (in pixels) and color of a node's label. Accept either dagviz's own <code>StyleConfig</code> values (the defaults: "sans-serif", "inherit", "currentColor") or their R equivalents directly.
label_arrow_stroke, label_arrow_dash_array	The color and linetype of the leader line from a node to its label. <code>label_arrow_dash_array</code> accepts dagviz's SVG stroke-dasharray value (the default, "2") or an R linetype code (e.g. "22") directly.

arc_radius	The radius of an input arc.
bridge_color	The color of the white "bridge" drawn under a horizontal line where it crosses other edges.
minimal_width	How far the plot extends to the right of its labels. NULL (the default) ends the plot just past the longest label, measured from its text. A number is dagviz's rule instead: the plot is at least that wide in pixels (dagviz uses 500) and twice as wide as the right-most label's start, since an SVG can't measure its own text. The plot is then often much wider than the graph.
node_size_range, edge_width_range	The smallest and largest node radius and edge width, in pixels, when node_size and edge_width (see dag_layers()) draw them from a column. Rows and lanes are 2 * scale apart, and the plot ends scale pixels from the nodes' centers, so a radius may not exceed scale, nor a width scale / 2. NULL (the default) is from half to one and a half times node_radius, and from half to two and a half times edge_stroke_width, kept within those limits. The limit is for circles: a triangle or diamond reaches up to about 1.6 times further, as for node_radius, so the biggest of them in the first or last row may be cut off unless scale is raised. Neighboring rows are 2 * scale apart, so two nodes of the largest radius in them leave only 2 * scale - 2 * max px for the line between them (2 px by default), too little to see. A smaller maximum, or a larger scale, leaves room.

Value

A dag_style_metro object.

Examples

```
style_metro()
dag <- as_dag(data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")))
ggplot_dag(dag, style = style_metro(node_radius = 9, arc_radius = 12, scale = 12))
```

tar_dag	<i>Read the current targets pipeline as a dag</i>
---------	---

Description

A thin wrapper around `targets::tar_network()`: runs it, then converts its vertices/edges result with `as_dag()`, keeping type (which `ggplot_dag()` draws as the node shape, see `targets_node_shapes()`), status, description, seconds, bytes and branches.

Usage

```
tar_dag(targets_only = TRUE, envir = parent.frame(), ...)
```

Arguments

targets_only	Passed to <code>targets::tar_network()</code> ; TRUE (unlike <code>tar_network()</code> 's own default of FALSE) because imported functions and objects clutter a metro plot.
envir	Environment to run the target script in. Forwarded explicitly (rather than left to <code>tar_network()</code> 's own default) so it resolves to the caller's environment, not a <code>ggtargets</code> function frame.
...	Passed to <code>targets::tar_network()</code> : names, shortcut, allow, exclude, outdated, reporter, callr_function, callr_arguments, script, store. names/allow/exclude accept tidyselect helpers exactly as in <code>tar_visnetwork()</code> .

Details

description is the one free-form field targets has for a target, and can hold the stage it belongs to, for `dag_collapse()` to collapse the pipeline by: see "Collapsing a pipeline by stage" in `tar_ggplot()`.

Value

A dag (see `as_dag()`).

Examples

```
targets::tar_dir({ # doesn't pollute the working directory
  targets::tar_script(list(
    targets::tar_target(x, 1),
    targets::tar_target(y, x + 1)
  ))
  ggplot_dag(tar_dag())
})
```

tar_ggplot

*Plot the current **targets** pipeline***Description**

The headline one-call function: `tar_dag()`, then `dag_collapse()` if you give a collapse_by, then `dag_layout()` and `ggplot_dag()`. Mirrors `targets::tar_visnetwork()`'s filtering arguments.

Usage

```
tar_ggplot(
  style = style_metro(),
  node_color = NULL,
  edge_color = "source",
  palette = metro_palette,
  node_shape = NULL,
```

```

    shapes = targets_node_shapes(),
    node_size = NULL,
    edge_width = NULL,
    collapse_by = NULL,
    fixed_size = TRUE,
    zoom = 1,
    targets_only = TRUE,
    envir = parent.frame(),
    ...
  )

```

Arguments

`style`, `node_color`, `edge_color`, `palette`, `node_shape`, `shapes`, `node_size`, `edge_width`

Passed to `dag_layout()/ggplot_dag()`. With `collapse_by`, the `n` of the collapsed graph's nodes and edges (the number of targets and of dependencies each stands for) is what `node_size = "n"` and `edge_width = "n"` draw.

`collapse_by` NULL (the default), or the name of a node column to collapse the pipeline by: the targets that share a value are merged into one node (see `dag_collapse()`). `tar_dag()` returns the columns type, status, description, seconds, bytes and branches, and description is the one to put a stage name in: see "Collapsing a pipeline by stage" above.

`fixed_size`, `zoom`

Passed to `ggplot_dag()`: by default, the plot has a fixed size so nodes, lines and text look the same in every pipeline.

`targets_only`, `envir`, ...

Passed to `tar_dag()`.

Details

To plot a pipeline in another folder: `script` and `store` locate the pipeline's files, but `tar_network()` runs `_targets.R` from the current working directory, and most pipelines use relative paths (e.g. `tar_source("R")`). Use `withr::with_dir("path/to/project", tar_ggplot())`.

As in `tar_visnetwork()`, the node shape shows its type: a circle for a regular target, a square for a dynamic-branching one, and triangles for imported functions and objects (only in the graph with `targets_only = FALSE`). A legend names the shapes when more than one is drawn. See `targets_node_shapes()`.

Value

A ggplot object.

Collapsing a pipeline by stage

`targets` has no tag or category field for a target, but it has one free-form text field, `description`, and `collapse_by = "description"` collapses the pipeline by it: the targets that share a description become one node, labeled with how many they are (ingest (4)), and the dependencies between

stages become one edge each (see `dag_collapse()`). With `node_size = "n"` and `edge_width = "n"`, the size of a node and the width of an edge show those counts.

A description can be set for each target, `tar_target(raw, read_data(), description = "ingest")`, or for a block of them with `tar_option_set()`, which applies to every target defined after it:

```
# _targets.R
library(targets)
tar_option_set(description = "ingest")
ingest <- list(tar_target(raw, read_data()), tar_target(clean, tidy(raw)))
tar_option_set(description = "model")
model <- list(tar_target(fit, fit_model(clean)))
tar_option_set(description = character(0)) # back to no description
c(ingest, model)

tar_ggplot(collapse_by = "description", node_size = "n", edge_width = "n")
```

Things to know:

- End a block with `tar_option_set(description = character(0))`. The option stays set for every target defined after it, including those in files loaded later with `tar_source()`.
- `tarchetypes::tar_map()` appends the mapped values to each description by default (`descriptions = tidyselect::everything()`), so the targets it makes would be stages of their own. Pass `descriptions = NULL` to keep the stage's description.
- The descriptions must match exactly: "ingest" and "Ingest " are two stages.
- A target with no description is a node of its own. A stage can't be named like such a target (an error: the two would be merged).
- Stages that depend on each other both ways ($A \rightarrow B \rightarrow A$) are an error that names the edges behind it.
- `description` is also the label of a target in `targets::tar_manifest()` and `targets::tar_visnetwork()`, so a short stage name is a good one, and a sentence is not.
- `node_color = "status"` colors a stage by its status when all its targets share one. A stage whose targets differ has no status, and is drawn gray with no legend entry, as is any node with a missing value in the `node_color` column.
- For metadata of your own, add a column to the nodes of `tar_dag()`'s result (its names can't be `row`, `col`, `x`, `y` or `color_index`), then call `dag_collapse()` and `ggplot_dag()` yourself.

Examples

```
# each call is its own tar_dir(), so each plot actually prints - inside
# one tar_dir({ ... }) block, only the last expression auto-prints
targets::tar_dir({
  targets::tar_script(list(
    targets::tar_target(x, 1),
    targets::tar_target(y, x + 1)
  ))
  tar_ggplot()
})
```

```

targets::tar_dir({
  targets::tar_script(list(
    targets::tar_target(x, 1),
    targets::tar_target(y, x + 1)
  ))
  tar_ggplot(node_color = "status", palette = targets_status_colors())
})

# the pipeline in stages, kept in the descriptions of its targets
targets::tar_dir({
  targets::tar_script(list(
    targets::tar_target(raw, 1, description = "ingest"),
    targets::tar_target(clean, raw + 1, description = "ingest"),
    targets::tar_target(fit, clean * 2, description = "model"),
    targets::tar_target(out, fit, description = "report")
  ))
  tar_ggplot(collapse_by = "description", node_size = "n", edge_width = "n")
})

```

targets_node_shapes *The **targets** node shapes*

Description

The shape `targets::tar_visnetwork()` draws for each vertex type, as a hard-coded copy - they belong to `targets`, not `ggtargets`, and are checked against the installed `targets` version in the package's own tests. It is the default shapes of `tar_ggplot()/ggplot_dag()/dag_layers()`, so a pipeline's nodes are drawn by type: a regular target (stem) is a circle, a dynamic-branching target (pattern) a square, and an imported function and object an up and a down triangle. Imports are only in the graph with `tar_ggplot(targets_only = FALSE)`.

Usage

```
targets_node_shapes()
```

Value

A named character vector of shape names, one per type. The shapes a node can take are "circle", "square", "diamond", "triangle" and "triangle_down".

Examples

```

targets_node_shapes()

dag <- as_dag(
  data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")),
  nodes = data.frame(
    name = c("a", "b", "c"),

```

```
      type = c("stem", "stem", "pattern")
    )
  )
  ggplot_dag(dag)
```

targets_status_colors *The **targets** status color palette*

Description

The colors `targets::tar_visnetwork()` uses for target status, as a hard-coded copy - they belong to `targets`, not `ggtargets`, and are checked against the installed `targets` version in the package's own tests, so a drift is caught rather than silently shipped. Fits the `palette` argument of `tar_ggplot()/ggplot_dag()/dag_layers()` directly, e.g. `tar_ggplot(node_color = "status", palette = targets_status_colors())`.

Usage

```
targets_status_colors()
```

Value

A named character vector of hex colors, one per status.

Examples

```
targets_status_colors()

dag <- as_dag(
  data.frame(from = c("a", "a", "b"), to = c("b", "c", "c")),
  nodes = data.frame(name = c("a", "b", "c"), status = c("uptodate", "outdated", "errored"))
)
ggplot_dag(dag, node_color = "status", palette = targets_status_colors())
```

Index

`as_dag`, [2](#)
`as_dag()`, [4](#), [7](#), [9](#), [14](#), [15](#)

`dag_collapse`, [3](#)
`dag_collapse()`, [6](#), [7](#), [15–17](#)
`dag_layers`, [5](#)
`dag_layers()`, [7–9](#), [11–14](#), [18](#), [19](#)
`dag_layout`, [7](#)
`dag_layout()`, [3](#), [5](#), [9](#), [15](#), [16](#)
`dag_size`, [8](#)
`dag_size()`, [10](#), [11](#)
`dag_theme`, [8](#)
`dag_theme()`, [5](#), [7](#)

`ggplot2::ggsave()`, [11](#)
`ggplot2::labs()`, [8](#)
`ggplot_dag`, [9](#)
`ggplot_dag()`, [2](#), [3](#), [5–8](#), [11](#), [14–19](#)
`ggsave_dag`, [11](#)
`ggsave_dag()`, [10](#)

`metro_palette`, [12](#)
`metro_palette()`, [6](#)

`style_metro`, [12](#)
`style_metro()`, [6](#), [7](#), [9](#)

`tar_dag`, [14](#)
`tar_dag()`, [2–4](#), [6](#), [7](#), [9](#), [15–17](#)
`tar_ggplot`, [15](#)
`tar_ggplot()`, [3](#), [4](#), [8](#), [15](#), [18](#), [19](#)
`targets::tar_manifest()`, [17](#)
`targets::tar_network()`, [2](#), [14](#), [15](#)
`targets::tar_visnetwork()`, [17](#)
`targets_node_shapes`, [18](#)
`targets_node_shapes()`, [6](#), [14](#), [16](#)
`targets_status_colors`, [19](#)