Workflow manager

using `targets`

Aurélien Ginolhac, Dpt. Life Sciences and Medecine

High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

HPC
UNIVERSITÉ DU
LUXEMBOURG
SCHOOL
2023

targets and companion package tarchetypes

A workflow manager for R

- Saving you time and stress
- Understand how it is implemented in `targets`
  - Define your `targets`
  - Connect `targets` to create the **dependencies**
  - Check **dependencies** with `visnetwork`
  - Embrace either
    - **Dynamic** branching
    - **Static** branching
  - Run **only** what needs to be executed
  - Embrace literate programming with `qmd` or `Rmd` docs
  - Bundle **dependencies** in a documents with `tar_render()`/`tar_quarto()`
  - Increase reproducibility with the package manager `renv`
- Example with RNA-seq data from **Wendkouni Nadège MINOUNGOU**

Folder structure

```
├── .git/
├── run.R
├── _targets.R
├── _targets/
├── Repro.Rproj
├── R
│   ├── functions.R
│   └── utils.R
├── run.R*
├── renv/
├── renv.lock
└── report.Rmd
```

Targets

- With `renv`. Snapshot your package environment (and restore! 😌)
- `_targets.R` is the only mandatory file
- Use a `R` sub-folder for functions, gets closer to a package
- `Rmarkdown`/`qmd` file allows to gather results in a report
- In a RStudio project
- Version tracked with git
- An executable `run.sh` allows to use Build Tools in RStudio

DatasauRus example, smart animation caching

This example is available at the `target_demos` repo(`hpc` branch)

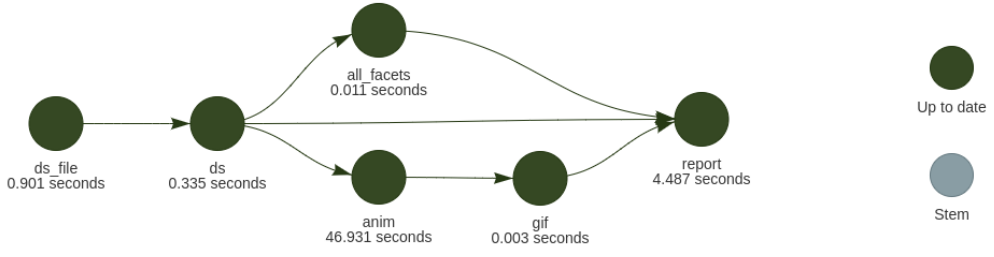
targets script `_targets_ds_fun1.R`

```
library(targets)
library(tarchetypes)
source("R/plotting.R")
# load the tidyverse quietly for each target
# which each runs in a fresh R session
tar_option_set(packages = "tidyverse")

list(
  # track if distant file has changed
  tar_url(ds_file, "https://raw.githubusercontent.com/jumpingri
  tar_target(ds, read_tsv(ds_file, show_col_types = FALSE)),
  tar_target(all_facets, facet_ds(ds)),
  # animation is worth caching  ~ 1 min
  tar_target(anim, anim_ds(ds),
             packages = c("ggplot2", "gganimate", "gifski")),
  tar_file(gif, {
    anim_save("ds.gif", animation = anim, title_frame = TRUE)
    # anim_save returns NULL, we need to get the file output pa
    "ds.gif"},
             packages = c("gganimate")),
  tar_render(report, "ds1.Rmd")
)
```

ımation code is presented as function, full code in `_targets_ds1.R`

Corresponding Directed Acyclic Graph

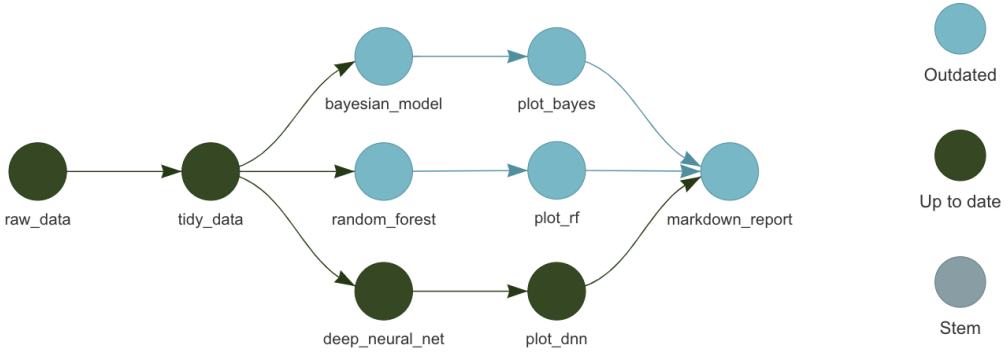



```
> tar_manifest()
# A tibble: 6 × 2
  name       command
  <chr>      <chr>
1 ds_file    "\"https://raw.gi[...]n/inst/extdata/DatasaurusDoz
2 ds         "read_tsv(ds_file, show_col_types = FALSE)"
3 anim       "anim_ds(ds)"
4 all_facets "facet_ds(ds)"
5 gif        "{\n     anim_save(\"ds.gif\", animation = anim, t
6 report     "tarchetypes::tar_render_run(path = \"ds1.Rmd\", a
```

Scalability and parallelization

- Scale-up with **dynamic** or **static** branching
- Parallelization on **HPC** using:
  - `tar_make_clustermq(workers = 3L)` `clustermq` by **Michael Schubert** based on ZeroMQ
  - `tar_make_future(workers = 3L)` `future` by **Henrik Bengtsson**
  - `tar_make()` (`mirai` by **Charlie Gao** based on NNG)

The last one is the recommended way, but just out!
Implemented in the recent `{crew}` package.



Source: **William Landau**: talk at Bayes Lund

Autoscaling of jobs on defined number of workers

All job on ~ 1 worker

```
library(targets)
controller <- crew::crew_controller_local(
  name = "my_controller",
  workers = 10,
  seconds_idle = 3
)
tar_option_set(controller = controller)
list(
  tar_target(x, seq_len(1000)),
  tar_target(y, x, pattern = map(x))
)
```

```
# after tar_make()
tar_crew()
#> # A tibble: 10 × 5
#>    controller    worker launches seconds targets
#>    <chr>          <int>    <int>   <dbl>   <int>
#>  1 my_controller      1        1   1.77      998
#>  2 my_controller      2        4   0.076       3
#>  3 my_controller      3        1   0           0
#>  4 my_controller      4        0   0           0
#>  5 my_controller      5        0   0           0
#>  6 my_controller      6        0   0           0
#>  7 my_controller      7        0   0           0
#>  8 my_controller      8        0   0           0
#>  9 my_controller      9        0   0           0
#> 10 my_controller     10        0   0           0
```

Better balance with 1 sec jobs

```
library(targets)
controller <- crew::crew_controller_local(
  name = "my_controller",
  workers = 10,
  seconds_idle = 3
)
tar_option_set(controller = controller)
list(
  tar_target(x, seq_len(1000)),
  tar_target(y, Sys.sleep(1), pattern = map(x)) # Run for 1 sec
)
```

```
# after tar_make()
tar_crew()
#> # A tibble: 10 × 5
#>    controller    worker launches seconds targets
#>    <chr>          <int>    <int>   <dbl>   <int>
#>  1 my_controller      1        1   103.      104
#>  2 my_controller      2        1   100.      100
#>  3 my_controller      3        1   100.      100
#>  4 my_controller      4        1   100.      100
#>  5 my_controller      5        1   100.      100
#>  6 my_controller      6        1   100.      100
#>  7 my_controller      7        1    99.4      99
#>  8 my_controller      8        1   100.      100
#>  9 my_controller      9        1    99.4      99
#> 10 my_controller     10        1    99.4      99
```

Literate programming

We recommend using it within a target and not the Target Markdown that overload the document.

```
---
title: "Datasaurus, one file"
output:
  html_document:
    theme: cosmo
date: "7th February 2023"
---

```{r}
#| label: setup
#| include: false
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(targets)
# We specify we will use the project ds_linear which
# correspond to the target file: _targets_ds_1.R and store `_ds_1/` (c
Sys.setenv(TAR_PROJECT = "ds_linear")
```

Literate programming using targets

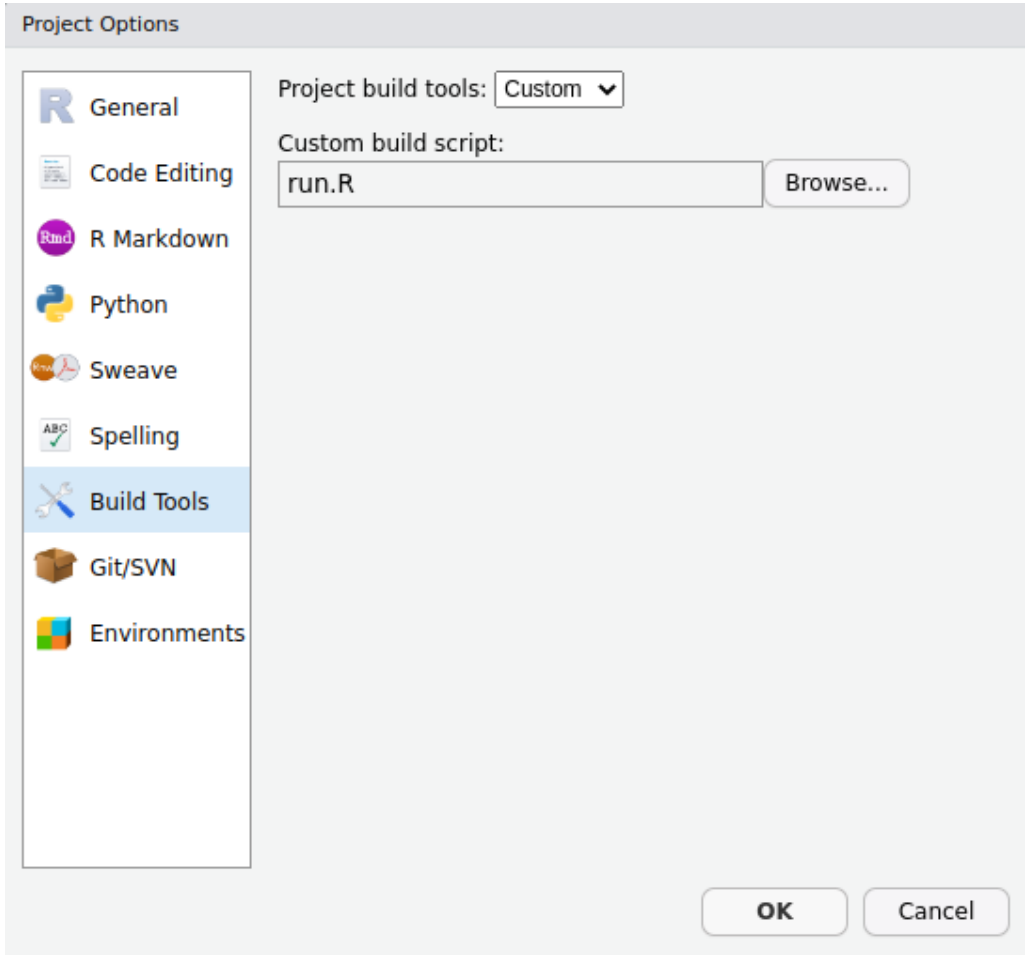
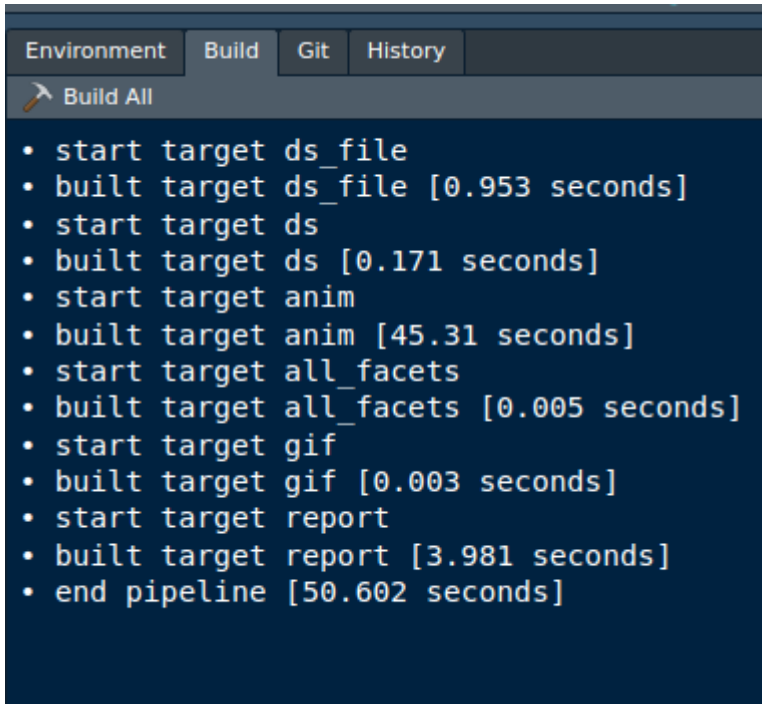
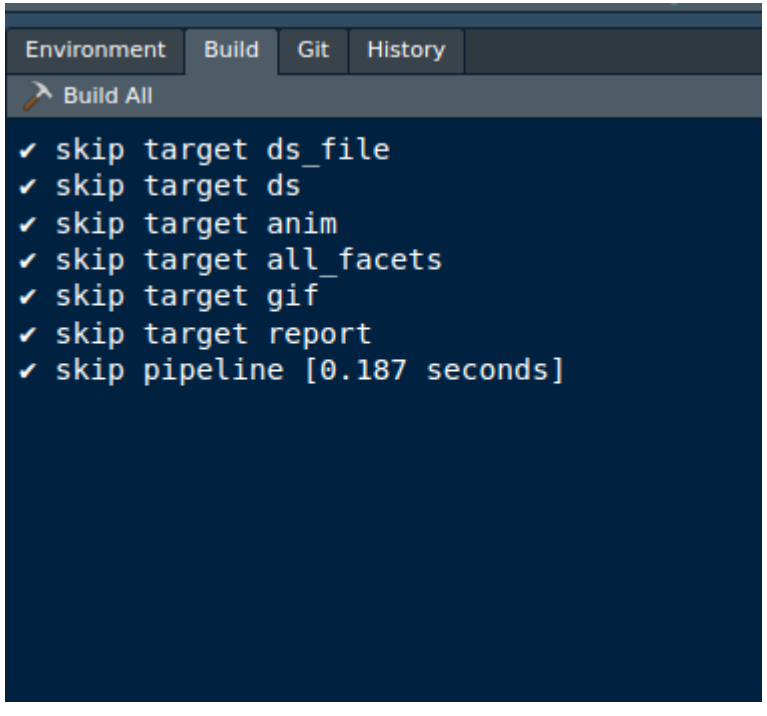
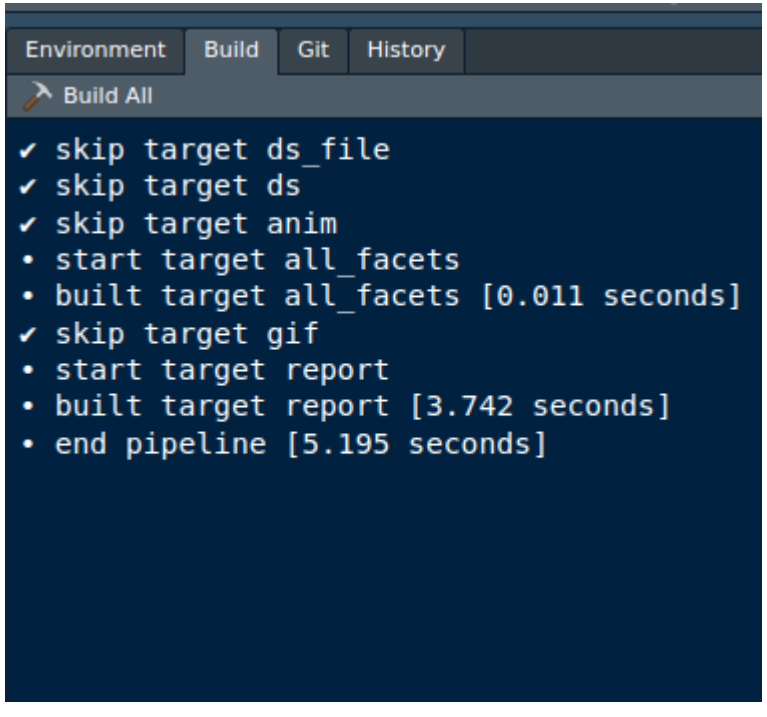
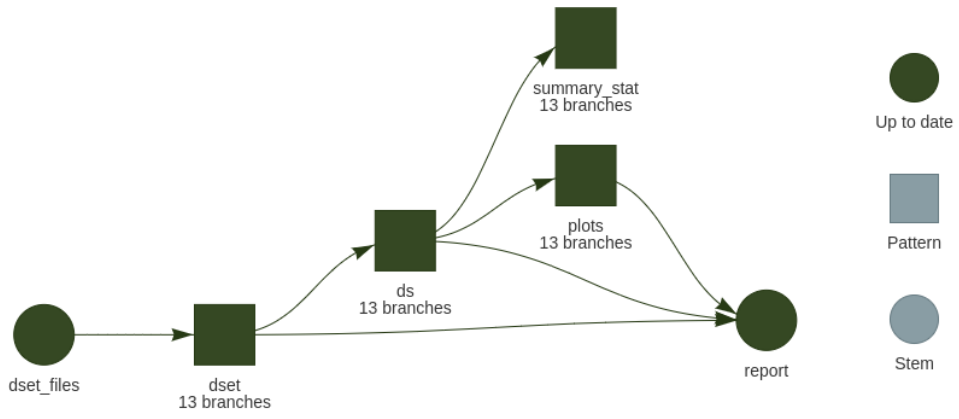
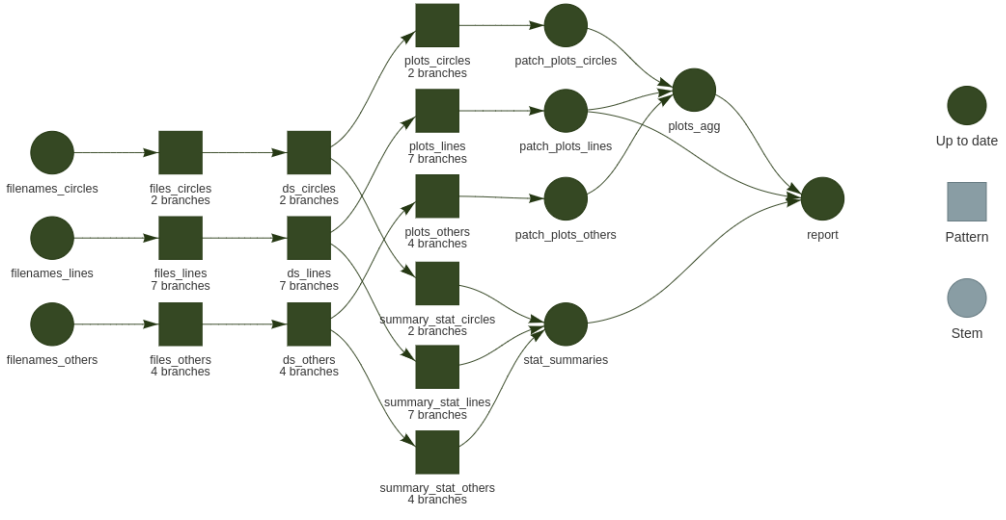
Read more in [William Landau](https://wlandau.github.io/about.html) `ta
[manual](https://books.ropensci.org/targets/literate-programming.html)

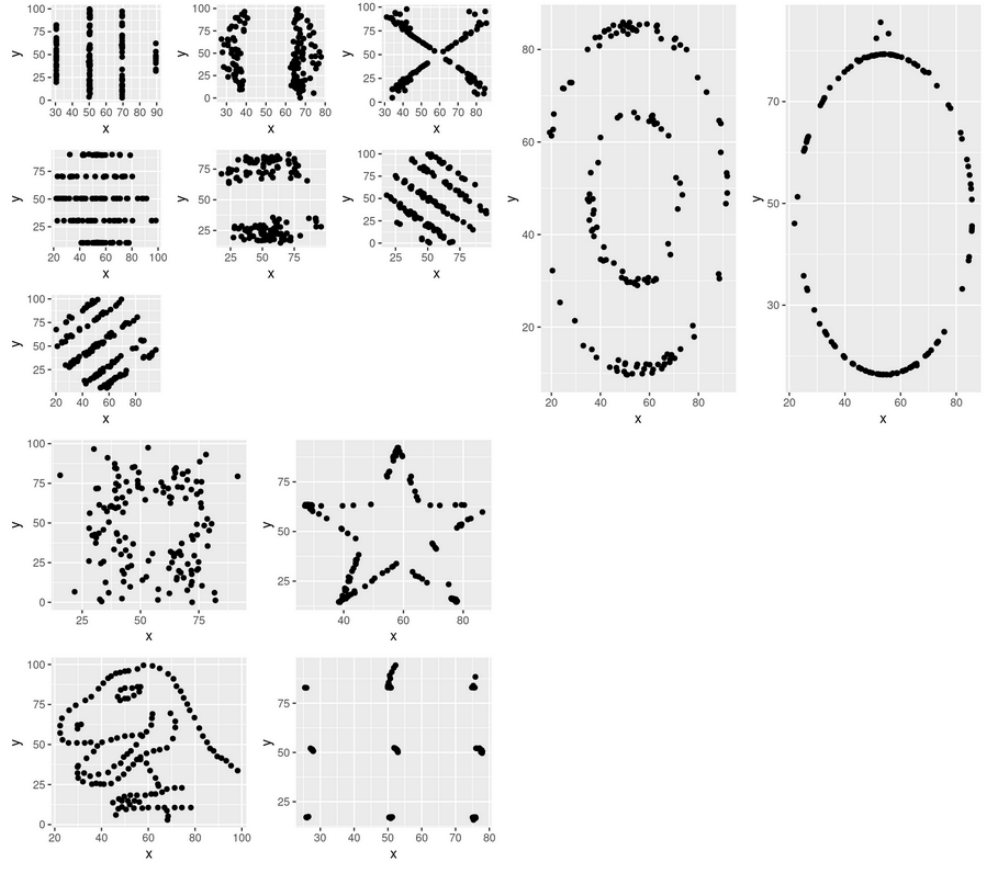
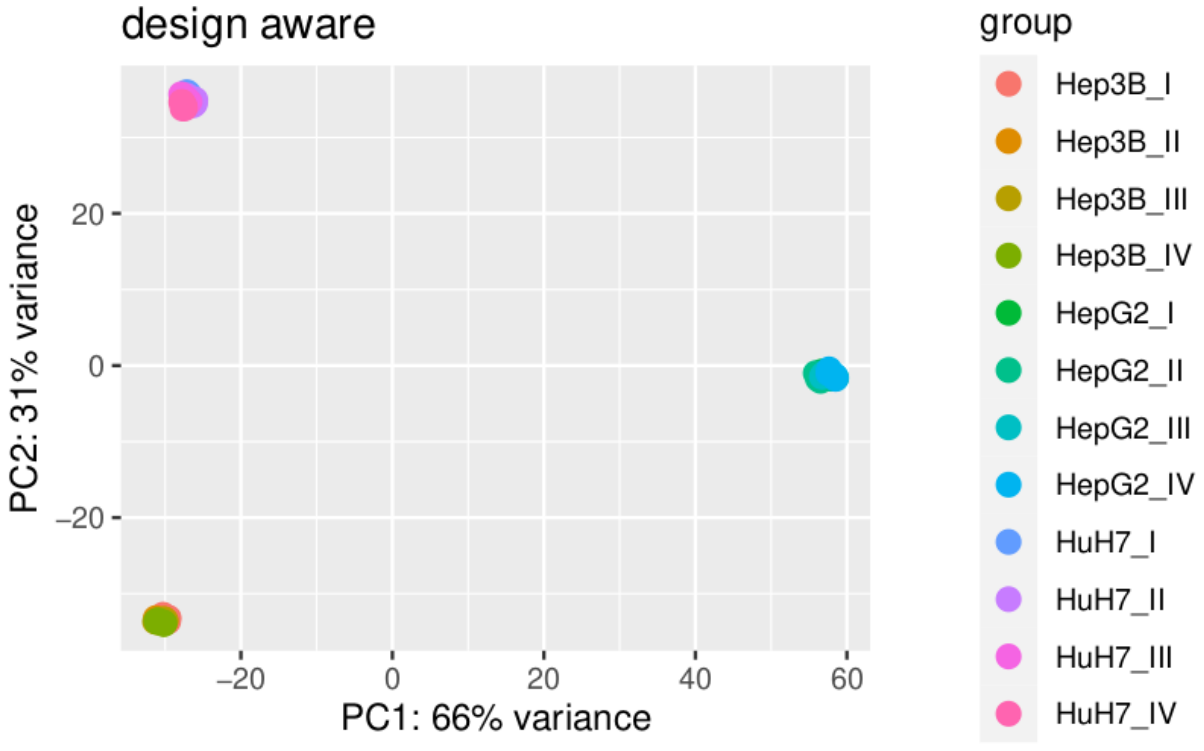
Dependencies of this Rmarkdown document are based on the parsing of the

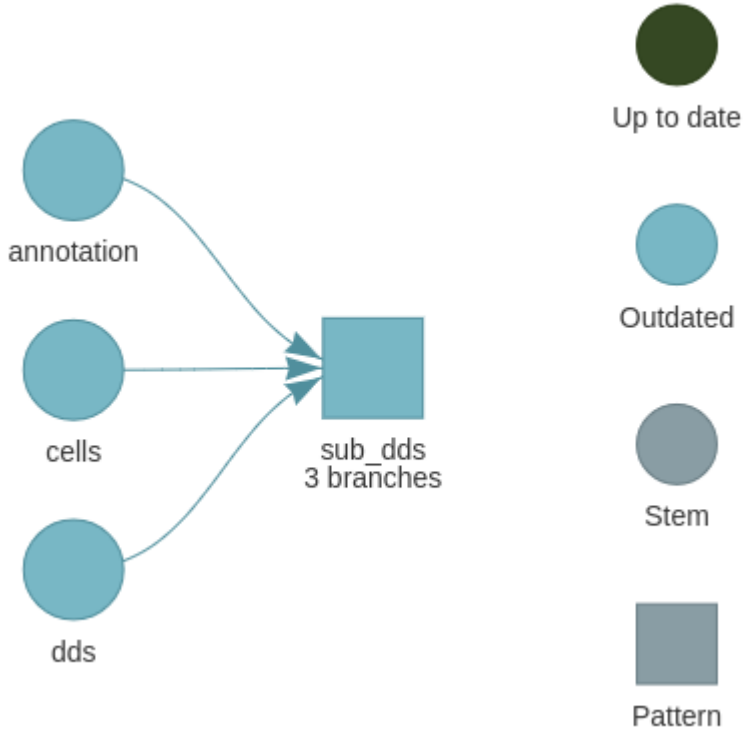
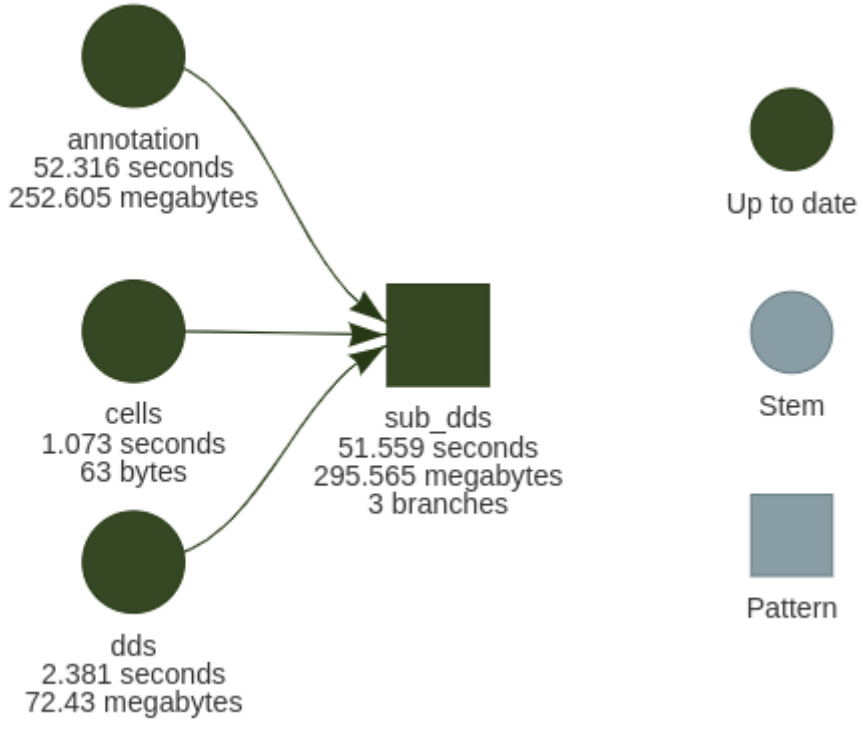
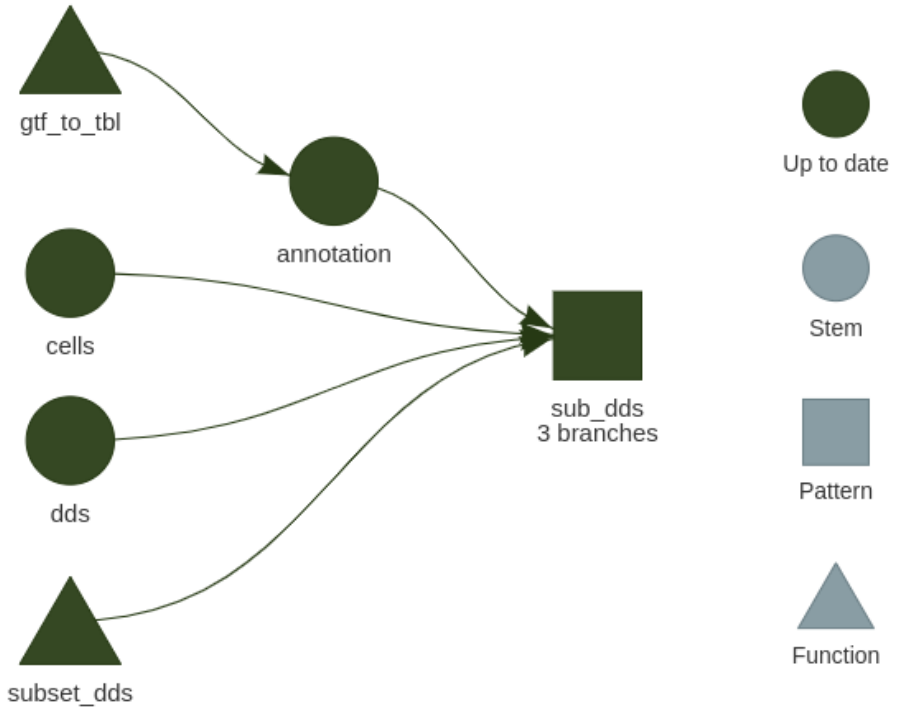
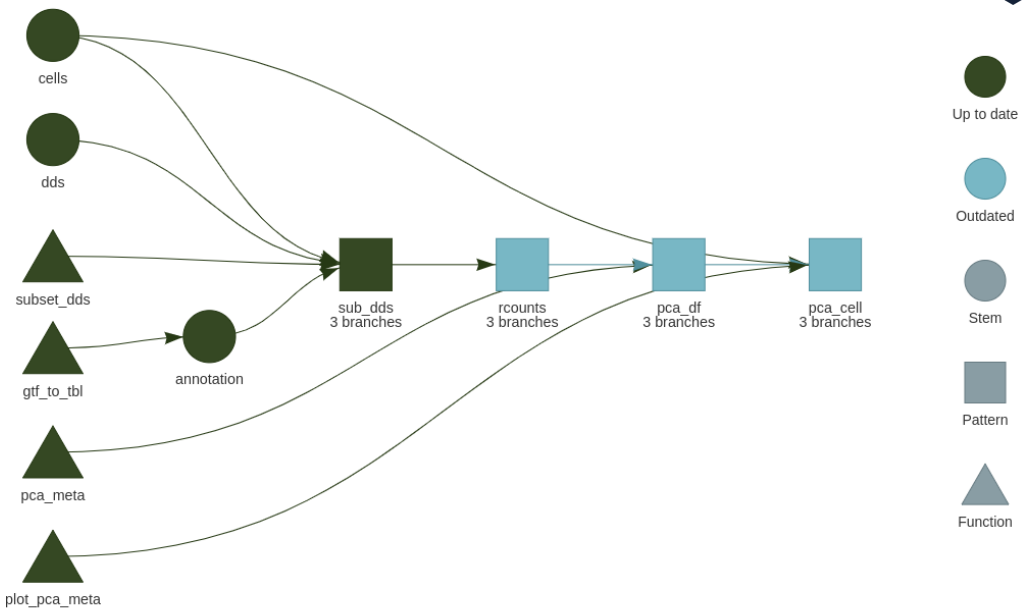
Tibble

- `tar_load()` read the target (by default a `rds` saved object) and lo
- `tar_read()` only reads the targets and returns it. No assignment is

```{r}
tar_load(ds)
ds
```

Compute summary statistics
```

Datasaurus, one file

7th February 2023

Literate programming using targets

Read more in William Landau `targets` manual.

Dependencies of this Rmarkdown document are based on the parsing of the `tar_read()` and `tar_load()` calls within it.

Tibble

- `tar_load()` read the target (by default a `rds` saved object) and load it in the GlobalEnv with the target name.
- `tar_read()` only reads the targets and returns it. No assignment is made.

```
tar_load(ds)
ds
```

```
## # A tibble: 1,846 × 3
##    dataset     x     y
##    <chr>   <dbl> <dbl>
##  1 dino     55.4  97.2
##  2 dino     51.5  96.0
##  3 dino     46.2  94.5
##  4 dino     42.8  91.4
##  5 dino     40.8  88.3
##  6 dino     38.7  84.9
##  7 dino     35.6  79.9
##  8 dino     33.1  77.6
##  9 dino     29.0  74.5
## 10 dino     26.2  71.4
## # … with 1,836 more rows
```

Compute summary statistics

```
ds |>
  group_by(dataset) |>
  summarise(across(c(x, y), list(mean = mean, sd = sd)))
```

Multi-projects in one folder

Like the `targets_demos` repo which has 4 projects

Config file: `_targets.yaml`

`targets` needs a R script and a store location

```
ds_linear:
  store: _ds_1
  script: _targets_ds_1.R
ds_fun_linear:
  store: _ds_fun1
  script: _targets_ds_fun1.R
ds_dynamic:
  store: _ds_2
  script: _targets_ds_2.R
ds_dynamic_crew:
  store: _ds_2_crew
  script: _targets_ds_2_crew.R
ds_static:
  store: _ds_3
  script: _targets_ds_3.R
  reporter_make: verbose_positives # do not display skipped tar
```

Usage

In your Rmd or console, one env variable to configure:

```
Sys.setenv(TAR_PROJECT = "ds_fun_linear")
```

Custom Building Tool

- Use the `run.R` as Custom file for Building

Tools > Projects Options > Custom



Running targets

- Useful shortcut **CTRL+SHIFT + B**



- Animation takes the most time

Re-running, same shortcut only what is needed

Without changes



Change in `facet_ds()`

```
facet_wrap(~ dataset, ncol = 4)
```
<- 3



Change that are comments are not invalidating a target

Dynamic branching

Often we start from multiple files

```
data
├── dset_10.tsv
├── dset_11.tsv
├── dset_12.tsv
├── dset_13.tsv
├── dset_1.tsv
├── dset_2.tsv
├── dset_3.tsv
├── dset_4.tsv
├── dset_5.tsv
├── dset_6.tsv
├── dset_7.tsv
├── dset_8.tsv
└── dset_9.tsv
```

And we want to apply the same treatment to each

Functional programming again, iteration for what's needed

Done by the `pattern = map()` keyword. Use `cross()` to combinations.

```
tar_target(ds, read_tsv(dset, show_col_types = FALSE),
           pattern = map(dset)),
```

Directly with `tar_files_input()`



Changing one input file

		@@ -16,11 +16,11 @@ slant_down	24.150491069
16	16	slant_down	28.6031956015	44.2237227
17	17	slant_down	20.9319996761	51.6462354868
18	18	slant_down	35.4355325251	34.9610381566
19	19	slant_down	44.3773828236	57.6847369215
20	20	slant_down	46.313692406	55.6855917703
21		slant_down	46.3860739231	51.0021645381
	21	slant_down	46.3860739231	51.9021645381
22	22	slant_down	42.4954400876	56.9036815315
23	23	slant_down	44.3005150328	58.1329592781
24	24	slant_down	46.9882833458	55.4206367313
25	25	slant_down	52.4215231623	47.867005488
26	26	slant_down	64.090998189	68.6509850107

Re-run only one file and downstream dependencies

```
✔ skip target dset_files
[...]
• start branch dset_3c9c9095
• built branch dset_3c9c9095 [0 seconds]
✔ skip branch dset_fe11a7b7
• built pattern dset
✔ skip branch ds_10a9d1c1
[...]
✔ skip branch ds_765f2283
• start branch ds_7f4fe112
• built branch ds_7f4fe112 [0.954 seconds]
✔ skip branch ds_dcc9ee18
• built pattern ds
✔ skip branch summary_stat_ad2f392a
[...]
• built branch summary_stat_3c976614 [0.01 seconds]
✔ skip branch summary_stat_90f7b2c1
• built pattern summary_stat
✔ skip branch plots_ad2f392a
[...]
✔ skip branch plots_1a450db7
• start branch plots_3c976614
• built branch plots_3c976614 [0.008 seconds]
✔ skip branch plots_90f7b2c1
• built pattern plots
• start target report
• built target report [4.104 seconds]
• end pipeline [5.659 seconds]
```

Automatic aggregation

For vectors/tibbles happens directly

```
> tar_read(ds)
## # A tibble: 1,846 × 3
##    dataset     x     y
##    <chr>   <dbl> <dbl>
##  1 away     32.3  61.4
##  2 away     53.4  26.2
##  3 away     63.9  30.8
```

Use branches for subsetting

```
> tar_read(ds, branches = 2L)
## # A tibble: 142 × 3
##    dataset     x     y
##    <chr>   <dbl> <dbl>
##  1 star     58.2  91.9
##  2 star     58.2  92.2
##  3 star     58.7  90.3
```

For plots, use `iteration = "list"`

```
tar_target(plots, ggplot(ds, aes(x, y)) +
             geom_point() +
             labs(title = unique(ds$dataset)),
           pattern = map(ds),
           iteration = "list"),
```

```
> tar_read(plots, branches = 2L)
## $plots_a55f1afc
```

Then this list can be used by `patchwork`

```
library(patchwork)
wrap_plots(tar_read(plots)) +
   plot_annotation(title = "13 datasets bundled with patchwork"
```

Static branching, with dynamic inside

Dynamic branch names are not meaningful, just hashes

Multi-folders input data

We still have multiple files per folder

```
circles
├── dset_2.tsv
└── dset_3.tsv
lines
├── dset_11.tsv
├── dset_12.tsv
├── dset_13.tsv
├── dset_6.tsv
├── dset_7.tsv
├── dset_8.tsv
└── dset_9.tsv
others
├── dset_10.tsv
├── dset_1.tsv
├── dset_4.tsv
└── dset_5.tsv
```

Dynamic within static, best of both worlds

More difficult to write with `tar_map()` (see example)

But meaningful names and combine when needed:



Use `tar_manifest()` to display exactly the command to be run

Parallel static branches and combine

From `_targets_ds_3.R`, static branches:

```
# Static branching with dynamic branching inside
values <- tibble(
  folders = c("lines", "circles", "others")
)

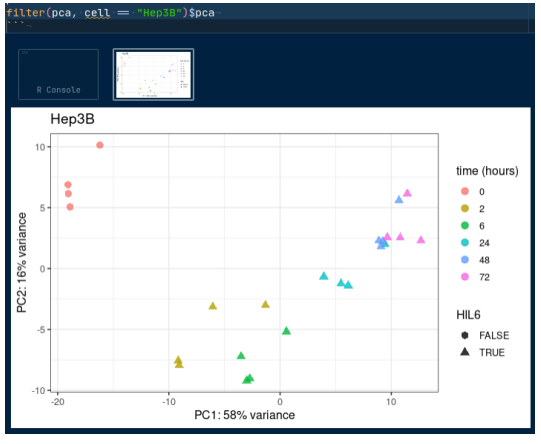
# tar_map() generates R expressions, and substitute the desired
mapped <- tar_map(
  values = values,
  names = "folders", # to avoid targets reporting "files_lines_
  tar_target(filenames, fs::dir_ls(folders, glob = "*tsv")),
  # filenames is not of format file, no checksum is done
  # we need a dynamic pattern at this step to read them dynamic
  tar_target(files, format = "file", filenames,
             pattern = map(filenames)),
  # Dynamic within static
  tar_target(ds, read_tsv(files, show_col_types = FALSE),
             pattern = map(files)),
  tar_target(summary_stat, summarise(ds, m_x = mean(x), m_y = m
             pattern = map(ds)),
  tar_target(plots, ggplot(ds, aes(x, y)) +
               geom_point(),
             pattern = map(ds),
             iteration = "list"),
  # Patchwork each group into one plot
  tar_target(patch_plots,
             wrap_plots(plots) +
               # Title the last bit of path_plots_{circles,line
               plot_annotation(title = stringr::str_split_i(tar
             packages = "patchwork")
)
```

Combining steps:

```
# We want to combined in one tibble the 3 tibble of summary sta
# Each of one them is actually composed of 2, 4 and 7 tibbles
stat_combined <- tar_combine(
  stat_summaries,
  mapped[["summary_stat"]],
  # Force evaluation using triple bang (!!!)
  command = dplyr::bind_rows(!!!.x, .id = "ds_type")
)
# And the plots now, a patchwork of patchwork
plot_combined <- tar_combine(
  plots_agg,
  mapped[["patch_plots"]],
  # Force evaluation of all patchwork plots again with triple b
  command = {wrap_plots(list(!!!.x), ncol = 2) +
               plot_annotation(title = "Master Saurus")},
  packages = "patchwork"
)
# Wrap all targets in one list
list(mapped,
     stat_combined,
     plot_combined,
     tar_render(report, "ds3.Rmd"))
```

Manifest and final plot

`tar_manifest()` (paged version in `ds3.Rmd`)

```
# A tibble: 21 × 3
name                  command
<chr>                 <chr>
filenames_circles     "fs::dir_ls(\"circles\", glob = \"*tsv\")"
filenames_others      "fs::dir_ls(\"others\", glob = \"*tsv\")"
filenames_lines       "fs::dir_ls(\"lines\", glob = \"*tsv\")"
files_circles         "filenames_circles"
files_others          "filenames_others"
files_lines           "filenames_lines"
ds_circles            "read_tsv(files_circles, show_col_types =
ds_others             "read_tsv(files_others, show_col_types = F
ds_lines              "read_tsv(files_lines, show_col_types = FA
summary_stat_circles "summarise(ds_circles, m_x = mean(x), m_y
plots_circles         "ggplot(ds_circles, aes(x, y)) + geom_poin
summary_stat_others   "summarise(ds_others, m_x = mean(x), m_y =
plots_others          "ggplot(ds_others, aes(x, y)) + geom_point
plots_lines           "ggplot(ds_lines, aes(x, y)) + geom_point(
summary_stat_lines    "summarise(ds_lines, m_x = mean(x), m_y =
patch_plots_circles   "wrap_plots(plots_circles) + plot_annotati
patch_plots_others    "wrap_plots(plots_others) + plot_annotatio
patch_plots_lines     "wrap_plots(plots_lines) + plot_annotation
stat_summaries        "dplyr::bind_rows(summary_stat_lines = sum
plots_agg             "wrap_plots(list(patch_plots_lines = patch
report                "tarchetypes::tar_render_run(path = \"ds3.
```

```
tar_read(plots_agg)
```

Master Saurus



RNA-seq real example

Example with RNA-seq data across 3 cell lines

PCA shows that differences between cells >> biological effect (roman numbers)

Solution: Split counts and metadata for each cell

Do we copy code 3 times?



Define targets = explicit dependencies

`_targets.R`, define 4 targets

Last `target` depends on the **3** upstreams

```
library(targets)
source("R/functions.R")
source("R/plotting.R")

list(
  tar_target(cells, c("HepG2", "HuH7", "Hep3B")),
  tar_qs(dds, read_rds(here::here("data", "all.rds")),
         packages = "DESeq2"),
  tar_fst_tbl(annotation, gtf_to_tbl(here::here("data",
                                                "gencode.v36.an
              packages = c("tibble", "rtracklayer")),
  tar_qs(sub_dds, subset_dds(dds,
                             filter(annotation, type == "gene")
                             .cell = cells),
         pattern = map(cells), # dynamic branching
         packages = c("DESeq2", "tidyverse"))
[...]
)
```

Dynamic branching makes dependencies easier to read.



gure from `tar_visnetwork()`

Running targets

```
• run target annotation
• run target cells
• run target dds
• run branch sub_dds_3078b1e0
         condition time_h
HepG2_I1   control      0
HepG2_I2      HIL6      2
using pre-existing size factors
estimating dispersions
gene-wise dispersion estimates: 2 workers
mean-dispersion relationship
final dispersion estimates, fitting model and testing: 2 worker
• run branch sub_dds_d05c5da7
        condition time_h
HuH7_I1   control      0
HuH7_I2      HIL6      2
using pre-existing size factors
estimating dispersions
gene-wise dispersion estimates: 2 workers
mean-dispersion relationship
final dispersion estimates, fitting model and testing: 2 worker
• run branch sub_dds_c60d7096
         condition time_h
Hep3B_I1   control      0
Hep3B_I2      HIL6      2
using pre-existing size factors
estimating dispersions
gene-wise dispersion estimates: 2 workers
mean-dispersion relationship
final dispersion estimates, fitting model and testing: 2 worker
 end pipeline
```

Options to display time and object sizes



Re-running

```
✓ skip target annotation
✓ skip target cells
✓ skip target dds
✓ skip branch sub_dds_3078b1e0
✓ skip branch sub_dds_d05c5da7
✓ skip branch sub_dds_c60d7096
✓ skip pipeline
```

All good, nothing to be done ✔️.

Actually `targets` tracks all objects and so functions

A more complete dependency graph shows **functions**



Let's add the PCA per cell type now

PCA, add 4 targets

Smaller targets avoid unnecessary re-running steps

```
[...]
tar_target(rcounts, vst(sub_dds, blind = TRUE),
           pattern = map(sub_dds),
           packages = c("DESeq2")),
tar_target(pca_df, pca_meta(rcounts),
           pattern = map(rcounts),
           packages = c("DESeq2", "tidyr", "dplyr")),
tar_target(pca_cell, tibble(cell = cells,
                            pca = list(plot_pca_meta(pca_df))),
           pattern = map(cells, pca_df),
           packages = c("ggplot2", "tibble"))
[...]
```



Translate into:

- For every cell data, compute regularized counts (`vst`: variance stabilization)
- For every regularized counts, compute PCA (`df`: data.frame, *i. e* a table)
- For every cell names / PCA tables, plot PCA in a table for easier labeling

PCA results

Running

Literate programming

load results in a Rmd doc, **separate** computation from report

- Plots in a list-column

```
## Split per cell types

```{r paged.print=FALSE}
pca <- tar_read("pca_cell")
pca
```

A tibble: 3 x 2
  cell pca
  <chr> <list>
1 HepG2 <gg>
2 HuH7 <gg>
3 Hep3B <gg>
```

How to display a plot



The full picture

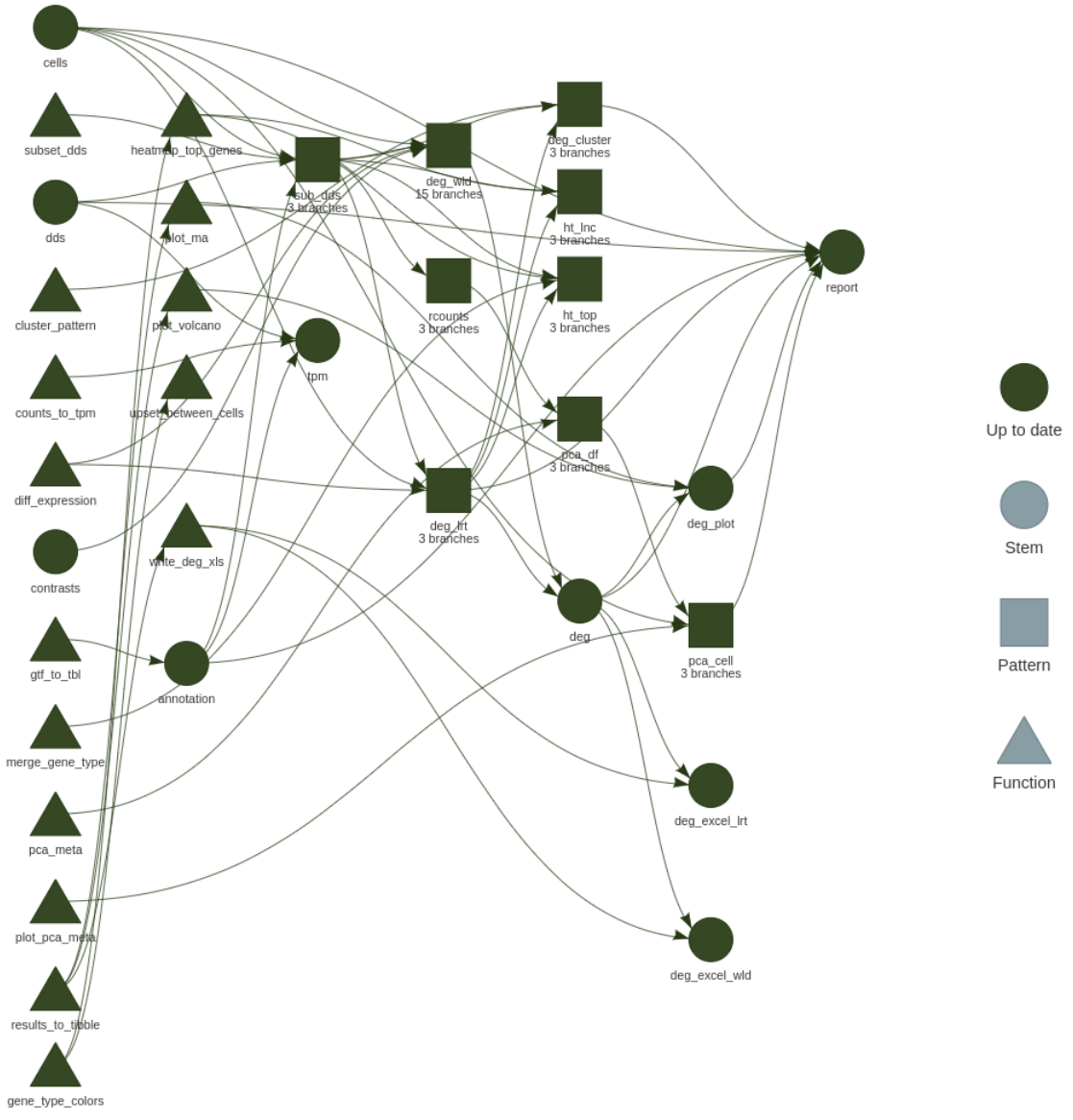
Adding step by step

desired analyses

Whole analysis takes 24 minutes and 4.54 seconds

> **Of course, someone has to remember the dependencies, it doesn't have to be you**

— *could be William Landau via **Jenny Bryan***



Is it worth the effort?

Yes

For you

- Autonomy
- Skills
- *Free* time
- Confidence over results
- Reproducibility
- Fun 🥳

Better project design

Thinking at what is a good `targets` helps tremendously the coding

> 1. Are large enough to subtract a decent amount of runtime when skipped.
> 2. Are small enough that some targets can be skipped even if others need to run.
> 3. Invoke no side effects (tar_target(format = "file") can save files.) 4.Return a single value that is
>   - Easy to understand and introspect.
>   - Meaningful to the project [...]
>
> — *William Landau*

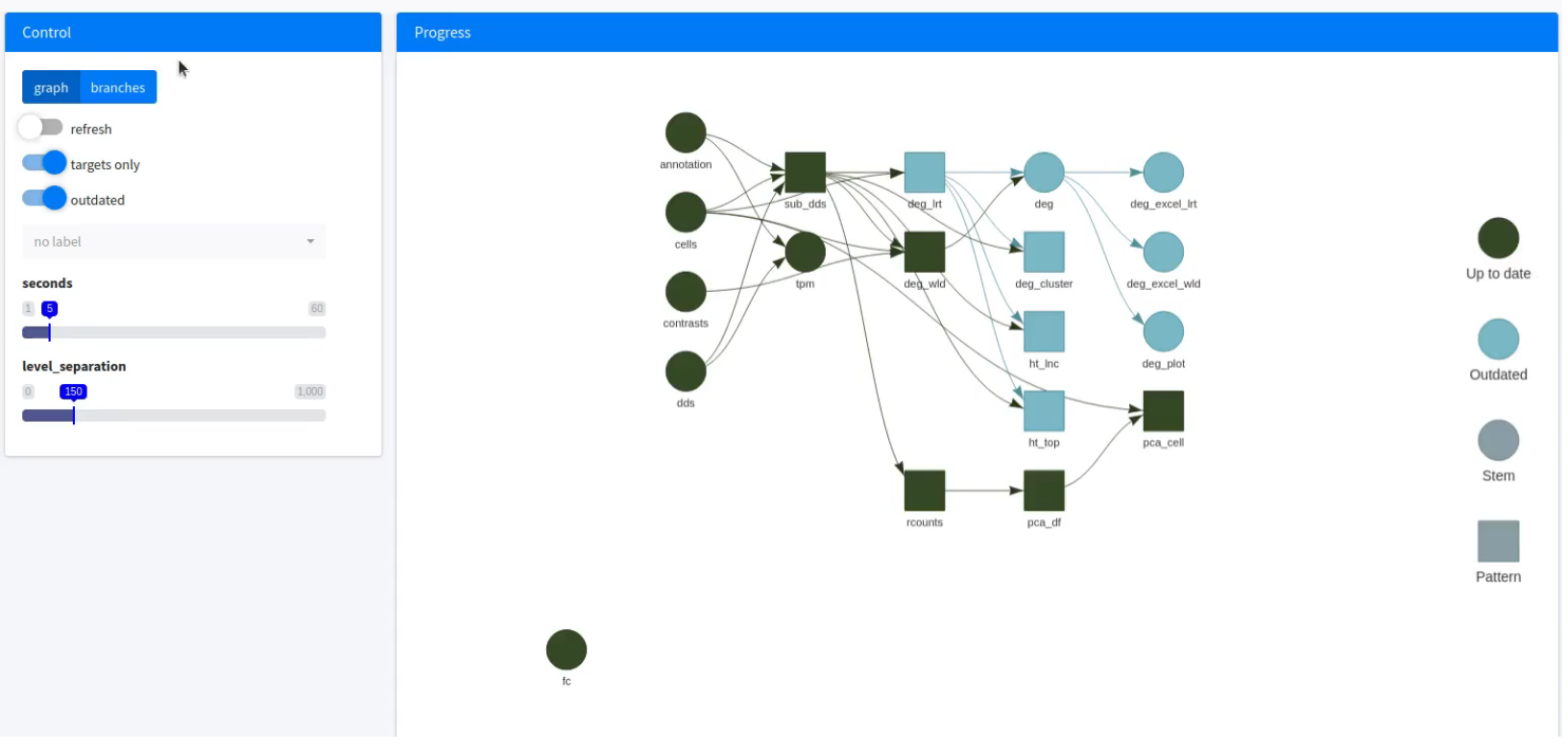
Reproducibility, thanks to `targets` and `renv` via `git`

```
> renv::history()
   commit         author_date      committer_date                                  subject
1e8dd2278 2021-02-23 15:29:57 2021-02-23 15:29:57                reformat creating config files
24c1222db 2021-02-15 17:07:01 2021-02-15 17:07:01    highlight gene type in the DEG patterns
326c8a726 2021-02-04 16:16:38 2021-02-04 16:16:38 cluster LRT genes by they dynamic patterns
4c6791796 2021-01-26 13:08:15 2021-01-26 13:08:15      gene types in upset plots for lengths
 865ee70b 2021-01-21 16:36:48 2021-01-21 16:37:08                             add upset plots
...]
```

Bonus: watch the pipeline running live

- `targets` events watched live 🎞
- Here, after changing a threshold in the LRT step
- `branches` can be monitored too
- 2 videos joined as I fixed an **error** at 1'42"
- Option to display functions (unset here)

`tar_watch()` shiny app from `targets`



Before we stop

Highlights

- `targets`, dependencies manager, re-run what's needed

Acknowledgments 🙏 👏

- **Eric Koncina** early adopter of `targets`
- **Wendkouni N. Minoungou** for the RNA-seq data
- **William Landau** main developer of `targets`

Further reading

- Documentation as bookdown by **Landau**
- Distributed computing using `{crew}`
- Targetopia **Landau** universe of targets-derived (stan/jags)
- Video from R Lille meetup by **William Landau**. June 2021 45"
- Video from Bayes Lund by **William Landau**. October 2021

Thank you for your attention!
Hope it was useful!