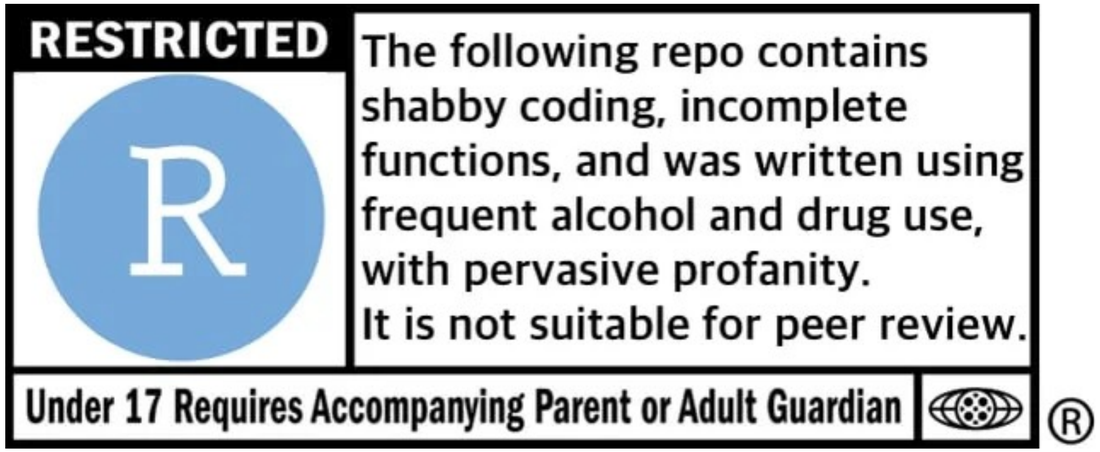
Working [smarter not harder] in R

A brief introduction to `targets` & friends



user manuals linked

AG Forslund || 28.06.22

Lightning round introduction

- Briefly share **your** biggest frustration with computational analysis or a recent struggle you had with R

Key concepts when thinking about reproducibility

- **Goal**: results that match the underlying code and data

- Project-oriented workflow
  - Put all files related to project in a single folder

- Dependencies
  - Files, variables you define, functions, packages

- Environments
  - Data structure that powers "scoping"
  - Collection of functions, variables, etc (dependencies!)

```
covid-opitz >> tree -L 1
.
├── README.md
├── _targets
├── _targets.R
├── code
├── covid-opitz.Rproj
├── data
├── figures
├── logfiles
├── make.R
├── meetings
├── miscellaneous
├── renv
├── renv.lock
├── results
└── rmd-notebooks
```

Root directory of project-oriented workflow

Tools that keep track of these concepts *for you*

- `renv` learns and records which packages (and versions) your code needs to run and stores it in a portable "lockfile" (renv.lock)

- `targets` learns which input files, computed objects and functions build on one another, so it knows what's affected when something gets changed

- `here` learns the file and folder structures within a project, and builds absolute file paths from the root that work if the project folder is moved

avoids:

```
> setwd("path/that/only/works/on/my/machine")
Error in setwd("path/that/only/works/on/my/machine") :
  cannot change working directory
```

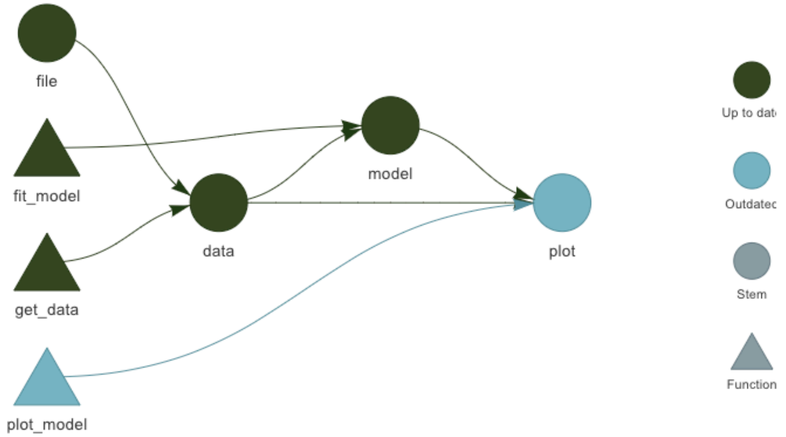
```
> renv::init()
* Initializing project ...
* Discovering package dependencies ... Done!
* Copying packages into the cache ... Done!
The following package(s) will be updated in the lockfile:

# CRAN ===============================
- renv   [* -> 0.12.5]

* Lockfile written to '~/projects/repro-demo/renv.lock'.

Restarting R session...

* Project '~/projects/repro-demo' loaded. [renv 0.12.5]
```



targets::all_you_need_know()

https://docs.ropensci.org/targets/reference/index.html

- `use_targets()` — loads necessary template scripts into your project directory

- `tar_target(variable.name, code)` — defines a target

- `tar_make()` — compiles code related to out-of-date targets (or specified targets) only
  - tar_make(variable.name), tar_make(c('data', 'plot'))
  - tar_make(contains('plot'))

- `tar_load()` — pull up a computed target for e.g. exploration in console or use in .Rmd
  - tar_load(contains('model'))

- `tar_outdated()` — lists which targets will be compiled with call to `tar_make()`
- `tar_visnetwork()` — visualize dependencies (interactive)

Caching in R Markdown

- Reference manual: https://yihui.org/knitr/options/

`cache = logical`

- Cached chunks are skipped, unless they have been modified
- Can be finicky in my experience

`cache.vars = character vector`

- Vector of variable names to save in the cache
- By default, all variables in all code chunks are cached

`cache.rebuild = logical`

- If true, always re-evaluate the chunk (rebuild this part of the cache)
- Could be conditional, e.g. cache.rebuild = !file.exists("some-file")

`cache.comments = logical`

- If false, changing comments in a code chunk will not invalidate variables in associated cache/not count as changes that cause a chunk to re-evaluate

Chunk label

Sets options globally

```
```{r setup, include = FALSE}

knitr::opts_chunk$set(
    cache = TRUE,
    cache.vars = c(),
    cache.rebuild = FALSE,
    cache.comments = FALSE
    )

    # load packages
    renv::activate()

    # load required targets
    targets::tar_load(……),
    store = here('_targets'))

```
```

Demonstration & implementation

- Quick look how I've implemented it in actual full-scale projects
  - https://git.bihealth.org/sxmorgan/gespic
  - https://git.bihealth.org/sxmorgan/covid-opitz

- Give it a try from scratch or start restructuring your own projects in remaining time

Possible things to demo with `targets`

- Running `targets` in the background with RStudio jobs function
- Making use of ## headings and dashed lines (create toggles/chunks to break up `_targets.R`)
  - Separating input files from raw files from whatever sensible groupings apply downstream
- When to abstract a function away and when to perform computation in `_targets.R` script
  - Pipes and ggplots in `_targets.R`
- Using `tarchetypes` to incorporate .Rmd reports into monitored dependency structure
  - `tar_render()` instead of `tar_target()`
- Debugging, `tar_invalidate()` when change isn't picked up for some reason
- Other aspects of project organization? Rmd headers?

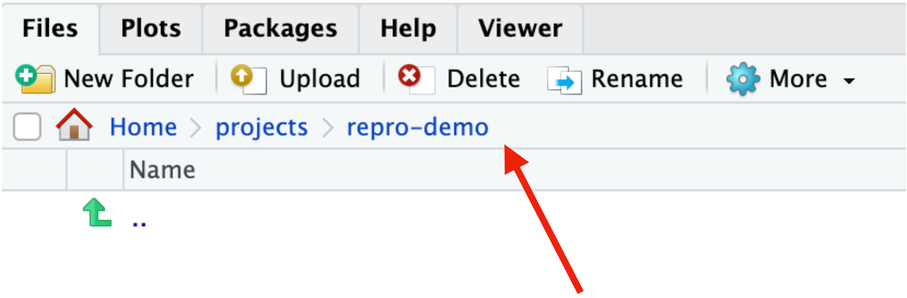
DIY: start a reproducible project in 5 steps

1. Install `targets`, `renv` and `here` from CRAN
2. Create a new project folder somewhere, call it **repro-demo**
3. Create an RStudio project and link it to this folder
4. Call `renv::init()` and start to build your "project-local" library
5. Call `targets::use_targets()` to pull up template files

* 2 and 3 can be combined by starting a new project, but I prefer to make a folder with my data first and then create a project

DIY: screenshots to follow along

2. Create a new project folder somewhere, call it **repro-demo**



I installed at /home/messex/projects/repro-demo on the VM

This will now be the **root** of your project folder, recognized by `targets` and `renv`

Root →
```
> here::here()
[1] "/home/messex/projects/repro-demo"
```

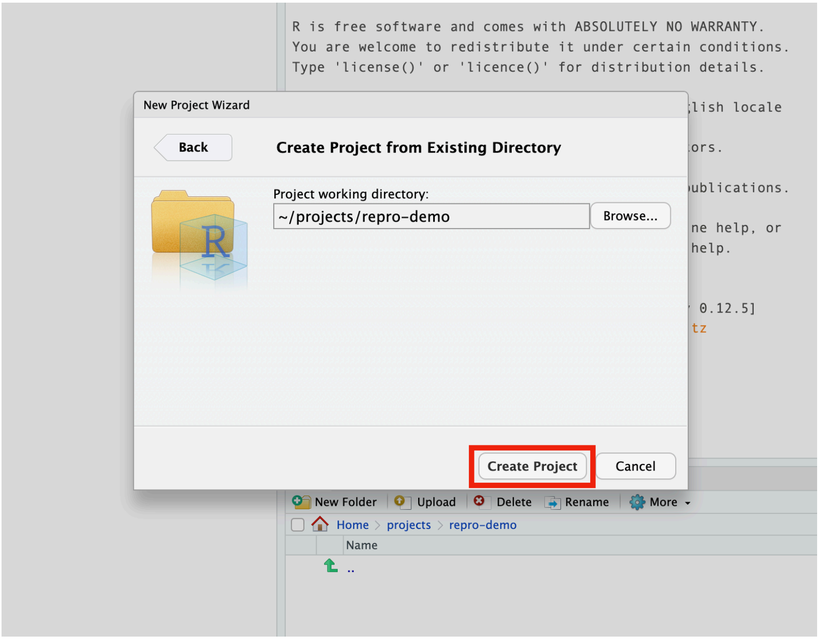
Path to renv folder →
```
> here::here('renv')
[1] "/home/messex/projects/repro-demo/renv"
```

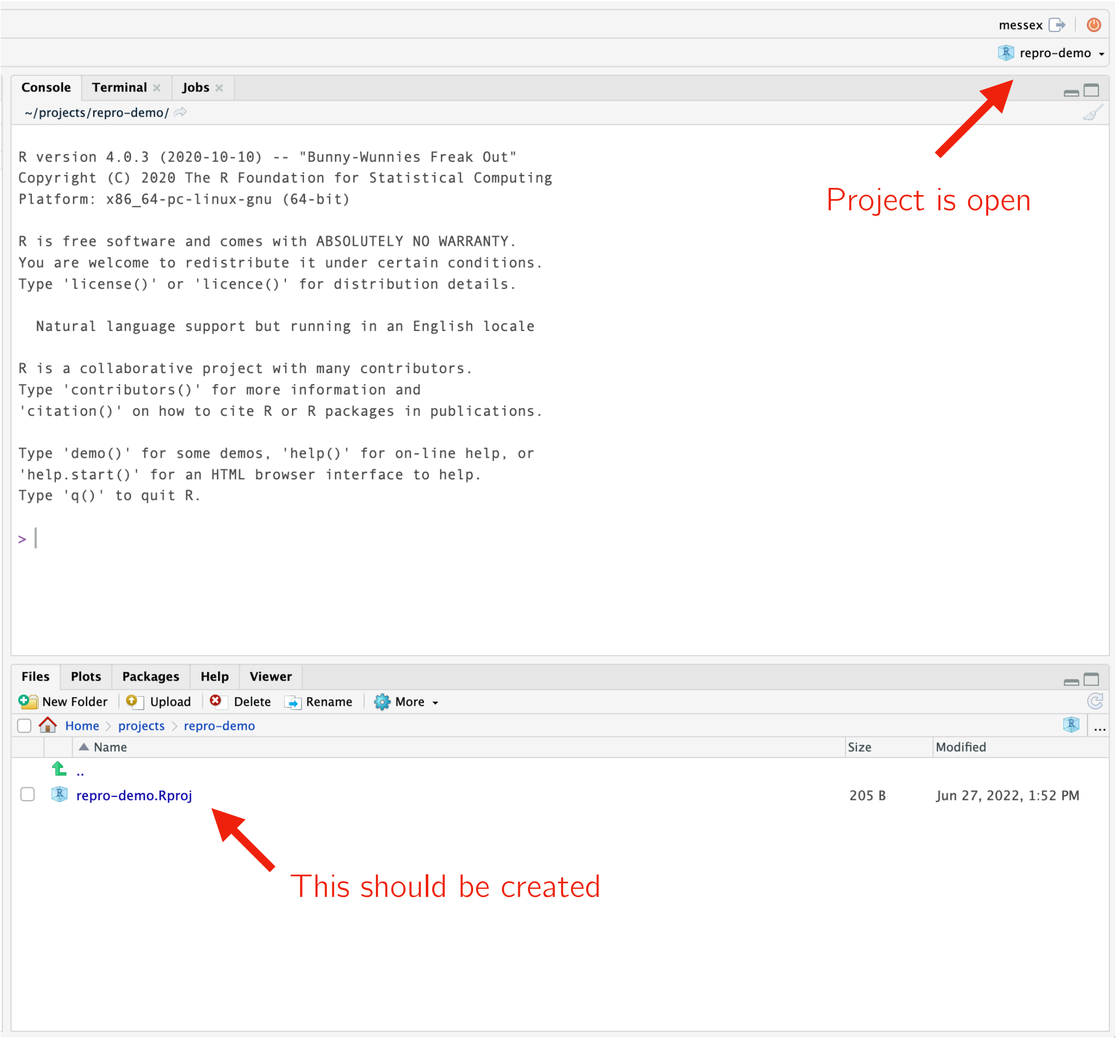
Path to existing file in renv folder →
```
> here::here('renv','activate.R')
[1] "/home/messex/projects/repro-demo/renv/activate.R"
```

Path to file in root folder that does not yet exist →
```
> here::here('new-file.pdf')
[1] "/home/messex/projects/repro-demo/new-file.pdf"
```

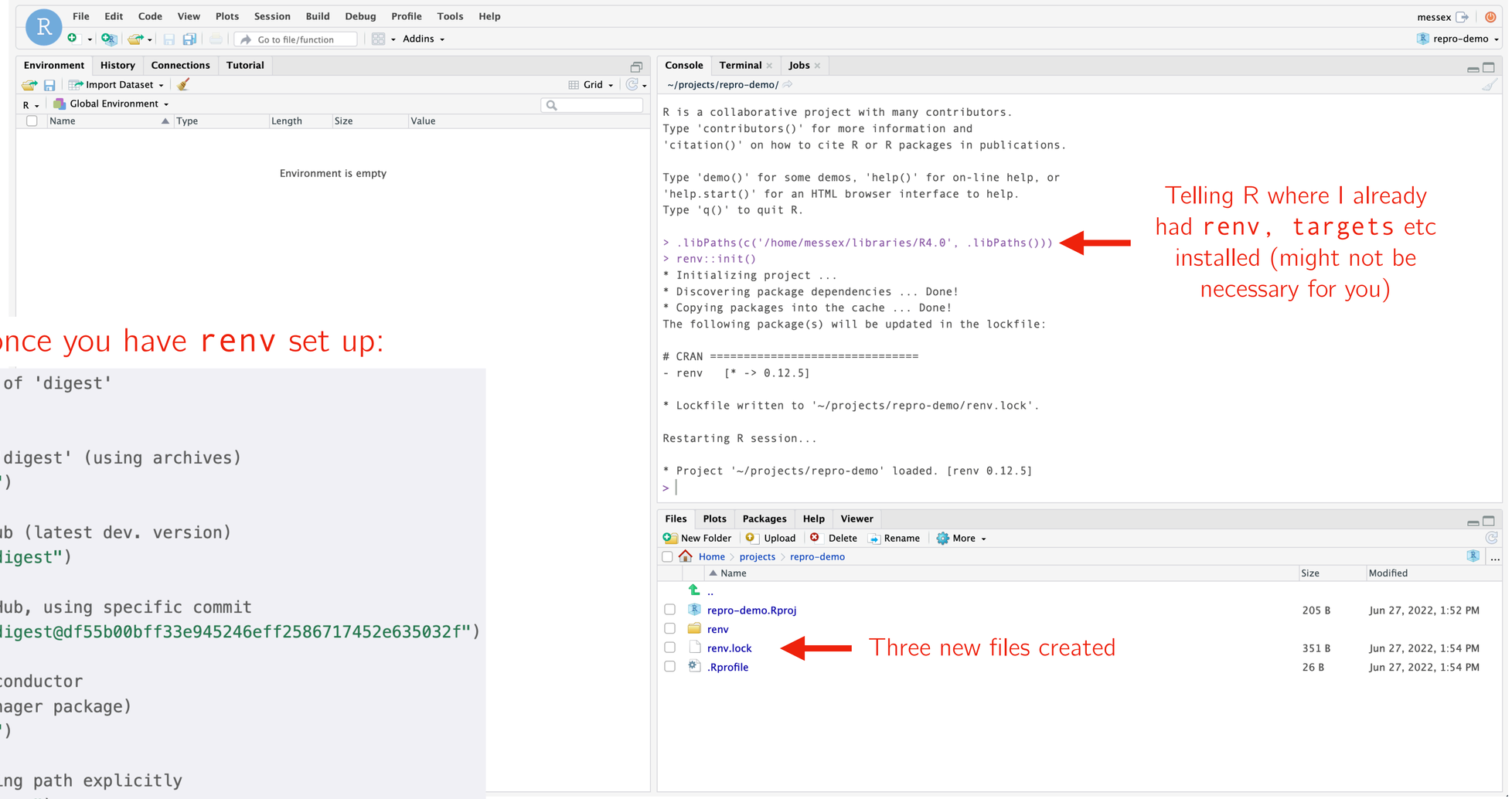
3. Create an RStudio project and link it to the repro-demo folder

Go to File —> New Project...





4. Call `renv::init()` and start to build your project-local library



Installing packages once you have `renv` set up:

```
# install the latest version of 'digest'
renv::install("digest")

# install an old version of 'digest' (using archives)
renv::install("digest@0.6.18")

# install 'digest' from GitHub (latest dev. version)
renv::install("eddelbuettel/digest")

# install a package from GitHub, using specific commit
renv::install("eddelbuettel/digest@df55b00bff33e945246eff2586717452e635032f")

# install a package from Bioconductor
# (note: requires the BiocManager package)
renv::install("bioc::Biobase")

# install a package, specifying path explicitly
renv::install("~/path/to/package")
```

5. Call `targets::use_targets()` to pull up template files

Don't rename!

_targets.R

```
# Created by use_targets().
# Follow the comments below to fill in this target script.
# Then follow the manual to check and run the pipeline:
#   https://books.ropensci.org/targets/walkthrough.html#inspect-the-pipeline # nolint

# Load packages required to define the pipeline:
library(targets)
# library(tarchetypes) # Load other packages as needed. # nolint

# Set target options:
tar_option_set(
  packages = c("tibble"), # packages that your targets need to run
  format = "rds" # default storage format
  # Set other options as needed.
)

# tar_make_clustermq() configuration (okay to leave alone):
options(clustermq.scheduler = "multicore")

# tar_make_future() configuration (okay to leave alone):
# Install packages {{future}}, {{future.callr}}, and {{future.batchtools}} to allow use_

# Load the R scripts with your custom functions:
lapply(list.files("R", full.names = TRUE, recursive = TRUE), source)
# source("other_functions.R") # Source other scripts as needed. # nolint

# Replace the target list below with your own:
list(
  tar_target(
    name = data,
    command = tibble(x = rnorm(100), y = rnorm(100))
#   format = "feather" # efficient storage of large data frames # nolint
  ),
  tar_target(
    name = model,
    command = coefficients(lm(y ~ x, data = data))
  )
)
```

```
> targets::use_targets()
✖ Install packages {{future}}, {{future.callr}}, and {{future.batchtools}} to allow use_targets() to configure tar_make_futu
re() options.
• Writing file "_targets.R".
• Writing file "run.R".
• Writing file "run.sh".
Error:
! The package `usethis` is required.
> renv::install('usethis')
Installing clipr [0.8.0] ...
        OK [linked cache]
Installing curl [4.3.2] ...
        OK [linked cache]
Installing desc [1.4.1] ...
        OK [linked cache]
Installing fs [1.5.2] ...
        OK [linked cache]
Installing sys [3.4] ...
        OK [linked cache]
Installing askpass [1.1] ...
        OK [linked cache]
Installing openssl [2.0.2] ...
        OK [linked cache]
Installing jsonlite [1.8.0] ...
        OK [linked cache]
Installing credentials [1.3.2] ...
        OK [linked cache]
Installing rstudioapi [0.13] ...
```

I had to install `usethis` package first

Name	Size	Modified
.Rprofile	26 B	Jun 27, 2022, 1:54 PM
renv		
renv.lock	351 B	Jun 27, 2022, 1:54 PM
repro-demo.Rproj	205 B	Jun 27, 2022, 1:54 PM
_targets.R	1.3 KB	Jun 27, 2022, 3:34 PM
run.R	294 B	Jun 27, 2022, 3:34 PM
run.sh	236 B	Jun 27, 2022, 3:34 PM

Three more new files created