

Package: hvtiRtemplates (via r-universe)

October 23, 2026

Type Package

Title Analysis Job Templates for the HVTI CORR Group

Version 1.4.2

Date 2026-10-20

Maintainer John Ehrlinger <john.ehrlinger@gmail.com>

License GPL (>= 3)

Encoding UTF-8

URL <https://github.com/ehrlinger/hvtiRtemplates>,
<https://ehrlinger.github.io/hvtiRtemplates/>

BugReports <https://github.com/ehrlinger/hvtiRtemplates/issues>

Description Versioned analysis job templates for the Cardiovascular Outcomes, Registries and Research (CORR) group within the Heart, Vascular and Thoracic Institute (HVTI) at the Cleveland Clinic. Provides the analysis-prefix taxonomy the group organizes jobs by, and resolves a template name to an installed file, so that a study binds to a versioned template rather than to a copy.

Depends R (>= 4.4.0)

Imports digest, hvtiRutilities (>= 1.5.7), jsonlite, openssl, ps, rlang, stats, yaml

Remotes ehrlinger/hvtiRutilities, ehrlinger/hvtiRtables, ehrlinger/hvtiRdatabuild, ehrlinger/hvtiRbootstrap, ehrlinger/hvtiRlifetables, ehrlinger/hvtiPlotR, ehrlinger/TemporalHazard, ehrlinger/ggRandomForests, ehrlinger/hvtiRpropensity, ehrlinger/ggBoostedTrees, boostmtree=ehrlinger/boostmtree_src/boostmtree@v2.0.8-ccf

Suggests arrow, bit64, boostmtree (>= 2.0.8), broom, cluster (>= 2.1.8), fs, ggBoostedTrees (>= 0.9.1), ggRandomForests (>= 4.0.0), ggiraph (>= 0.9.5), ggplot2, haven, hvtiPlotR (>= 2.8.9), hvtiRtables (>= 1.0.11), hvtiRdatabuild (>= 0.3.0), hvtiRbootstrap (>= 0.9.3), hvtiRlifetables (>= 0.1.2),

hvtiRpropensity ($\geq 0.2.0$), knitr, MASS, mice ($\geq 3.19.0$),
 nnet, numDeriv, officer, patchwork ($\geq 1.1.0$), quarto, ragg ($\geq$
 1.2.1), randomForestSRC ($\geq 3.7.0$), survival, TemporalHazard
 ($\geq 1.2.12$), testthat ($\geq 3.0.0$), varPro ($\geq 3.2.0$), withr,
 xml2

VignetteBuilder quarto

SystemRequirements Quarto command line tool
 (<<https://github.com/quarto-dev/quarto-cli>>)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs make libicu-dev libssl-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-10-23 01:30:28 UTC

RemoteUrl <https://github.com/ehrlinger/hvtiRtemplates>

RemoteRef HEAD

RemoteSha 19b893319bcd4f6b18d3ad8eb6faf84f6a0b0e6b

Contents

add_job	3
build_cohort	5
cancel_job	6
guard_long_fit	8
job_status	8
life_table	9
migrate_job	11
open_job	13
read_job_data	15
render_job	19
stop_here	20
submit_job	21
template_catalog	22
template_list	23
template_path	24
upgrade_job	25

Index 27

add_job	<i>Scaffold a new analysis job from a template</i>
---------	--

Description

Copies a supported job template into the taxonomy folder it belongs to, named `<prefix>[.<qualifier>].<subject>.<type>` so a study's jobs sort by template, then subject, then type. Refuses to overwrite an existing job: a job file accumulates a study's edits, and silently replacing one would discard them.

Usage

```
add_job(prefix, subject, type, dir = NULL, qualifier = NULL)
```

Arguments

prefix	Job type: one of the prefixes reported by template_list , or a template's full name as reported in its name column, e.g. "dc.trends"; the dash spelling "dc-trends" is accepted too. A full name carries the qualifier, so qualifier must then be left NULL.
subject	Grouping topic for the job set, e.g. "death" or "cohort". A subject names a statistical endpoint only when the job analyses one. Your choice: there is no list of valid values, and every job of one analysis should share it (see Details). Must match <code>^[A-Za-z0-9_]+\$</code> : . separates the filename's fields, and - separated them in names before 2026-10, so neither may appear here.
type	The analysis type the job's set belongs to, e.g. "hz". Your choice, like subject, and shared the same way. Must match <code>^[A-Za-z0-9_]+\$</code> , for the same reason as subject.
dir	The study root to write into. The taxonomy folder beneath it is created if it does not exist. NULL, the default, finds the study root by walking up from the working directory, and is an error outside a study, so a job is never written somewhere that is not a study.
qualifier	Job type within the prefix, e.g. "trends" for dp. Required only where a prefix carries more than one template; omitting it there is an error naming the choices, never a silent pick. Restricted to <code>[A-Za-z0-9_]+</code> , because . separates the filename's fields.

Details

Packages. When the study is a renv project, the packages the template attaches are installed into its library and recorded in `renv.lock` before the job is written; if they cannot be had, nothing is written. Outside a renv project nothing is installed.

What each argument decides. `prefix` chooses the template, together with `qualifier` where a prefix carries several job types. `subject` and `type` are yours to choose. The catalog holds no list of valid values for either, only the rule that each matches `^[A-Za-z0-9_]+$`. They are more than a filename, though. Together they name the job's set, and the set is used in four places:

- the job's filename, `<prefix>[.<qualifier>].<subject>.<type>.qmd`;
- the job's own SUBJECT and TYPE lines, which `add_job()` rewrites to your values;
- the render, which stops when the filename and those two lines disagree, so a job renamed by hand cannot quietly write its results into another set;
- the folder the job saves its results in, a `<subject>-<type>` folder under the study's estimates (and, for some templates, graphs), which is where the next job in the chain looks for them.

The last of these is the one that bites. `hm` reads the `hz.rds` that `hz` saved, and finds it only when both jobs carry the same subject and type. Give every job of one analysis the same pair, e.g. `subject = "death"`, `type = "hz"` for `ac`, `hz`, `hm` and `hp`, and give a different analysis a different pair. The `call` column of [template_list](#) shows each template's own example values, which are a starting point, not a requirement.

A job is identified by three or four fields. One or two come from the template, its prefix and, where the prefix carries several job types, its qualifier; two come from the caller. The pair (subject, type) names the **set** the job belongs to, and both are required. The subject is the grouping topic, not necessarily a statistical endpoint. An endpoint-driven job may use "death"; an endpoint-free job may use "cohort", "treatment", or "labs" without inventing an outcome. One subject can be analyzed by several methods, and the jobs those chains share would otherwise collide. A death-hazard set and a death random-forest-survival set both begin from the same life table, so keyed on the subject alone both would be written to one filename.

Scaffolding also installs the study's Quarto provenance hooks. Existing pre-render and post-render commands and unrelated project settings are preserved, while the provenance publisher is kept last. Repeated calls are idempotent.

A template whose job runs from a companion script also writes that script beside the job, from `inst/runners/<name>-runner.R`: today the bootstrap reports `bl`, `br`, `bc` and `bh`, whose runner screens and saves the bag the report reads. The runner is named `<prefix>[.<qualifier>].<subject>.<type>.runner.R`, gets the same SUBJECT and TYPE substitution, and is refused, like the job, if it already exists. Its study choices carry EDIT: markers for the author to work.

Names before 2026-10. Jobs were named `<subject>-<type>-<prefix>[.<qualifier>].qmd` until 2026-10. They keep that name and keep rendering; `add_job()` refuses to write a second copy of such a job under the new name, and [open_job](#) opens it.

A template the catalog marks deprecated still scaffolds, with a warning naming its replacement; see [template_catalog](#).

Renamed templates. `dp.eda` and `dp.trends` were renamed `dc.eda` and `dc.trends` on 2026-10-10. The old names still scaffold the renamed template, with a warning naming the new name, through the next release, and are removed in the one after. A `dc.trends` job is written to the descriptive folder, where a `dp.trends` job was written to graphs. A job scaffolded under the old name keeps it and keeps rendering; `add_job()` refuses to write a second copy of it under the new name, and [open_job](#) opens it.

Value

The job's path, invisibly. On any failure – including one after the copy, while substituting the set markers – no file is left behind, the runner included, so a returned path always names a complete, correctly-declared job.

See Also

[template_list](#), [template_path](#)

Examples

```
d <- file.path(tempdir(), "add-job-example")
invisible(hvtiRutilities::study_setup(
  d, study = "Example", study_tracker_id = 1L, renv = FALSE
))
add_job(prefix = "ac", subject = "death", type = "hz", dir = d)

# A qualified template by its full name, the form template_list()$call prints.
add_job("dc.gfup", subject = "cohort", type = "eda", dir = d)

# A template's name before a rename still works, and the warning names the
# new one.
tryCatch(add_job("dp.trends", subject = "cohort", type = "eda", dir = d),
  warning = conditionMessage)

# A job accumulates a study's edits, so an existing one is never overwritten.
try(add_job(prefix = "ac", subject = "death", type = "hz", dir = d))

list.files(d, pattern = "[.]qmd$", recursive = TRUE)
unlink(d, recursive = TRUE)
```

build_cohort	<i>Cut a study cohort from a master, recording who was dropped and why</i>
--------------	--

Description

The cohort step of a bd build job. It keeps the master rows a study is made of and returns, beside them, an attrition table: one row per step, with the rows each step removed and the patients left. A step is named by its reason, never by its rule's text, so the table is safe to print whatever a rule names.

Usage

```
build_cohort(data, exclude = NULL, cohort = NULL, join_by = NULL, id = "ccfid")
```

Arguments

data	A data frame read from the master.
exclude	NULL, or a list of rules written condition ~ "Reason", applied in order to the rows still kept. Each condition is evaluated against the data, with the rule's own environment for anything else it names, and must give TRUE or FALSE for every row. A row whose condition is missing is not excluded, as SAS if ... then delete and an hvtiRdatabuild analysis set treat it, and the number of such rows is counted. A downstream job's WHERE drops such rows instead.

cohort	NULL, or the path to a CSV cohort list, such as a REDCap export, carrying the join_by columns. Only master rows matched by a list row are kept. Dates are written YYYY-MM-DD. A list may carry exclude (1 to exclude) and reason columns; each reason becomes a step of its own. A list's reasons are printed in the report, so they must be categories (such as "Redo operation"), never free text, names or dates.
join_by	The columns that match a cohort-list row to a master row. Used only with cohort.
id	The patient identifier column, used to count patients.

Details

Identifiers are compared as text, so 100000 in the master matches "100000" in a list. No message names a data value: list rows that matched no master row are counted, never listed, and a rule that fails is named by its number and reason.

Value

A list:

- data, the kept rows, with the master's columns;
- attrition, a data frame with one row per step and the columns step, reason, rows_before, removed, missing_condition, rows_after and patients_after.

Examples

```
d <- data.frame(ccfid = c("S1", "S2", "S3", "S4"), age = c(17, 45, NA, 80),
               redo = c(0, 0, 0, 1))
cut <- build_cohort(d, exclude = list(age < 18 ~ "Under 18",
                                     redo == 1 ~ "Redo operation"))
# One row per step: rows removed, rows whose condition was missing and so
# kept (S3, with no age), and the patients left.
cut$attrition
cut$data
```

cancel_job

Stop a background job

Description

Stops a job [submit_job](#) started, and records the end the killed process could not: <job>.run.yml then reads cancelled, and <job>.run.progress is removed, so [job_status](#) stops reporting the run and the job can be submitted again.

Usage

```
cancel_job(  
  path,  
  grace = 30,  
  other_host = c("refuse", "abandon"),  
  quiet = FALSE  
)
```

Arguments

path	Character(1). The job's .qmd file.
grace	Numeric(1). Seconds to wait after SIGTERM before SIGKILL.
other_host	Character(1). For a job recorded running on another host, "refuse" (the default) stops with an error naming that host, and "abandon" records the run ended without signalling anything. Ignored for a job on this host.
quiet	Logical(1). FALSE prints the job's state afterwards, as job_status does.

Details

The render and every process it started are sent SIGTERM, given grace seconds to exit, then sent SIGKILL. They are found by a marker `submit_job` puts in the render's environment, which reaches processes Quarto starts in sessions of their own, and by descent from the recorded process, which is signalled only while it is still the one the status file records: it must carry the run's marker or, recorded without one, match its creation time to the second, so a pid since reused by another process is left alone, and an identity in doubt is refused. The job is scanned afresh until no process of it remains, so one started while it shuts down is stopped too. A run recorded without a creation time, such as a render started without `submit_job`, is refused rather than signalled by its pid alone.

A process can only be checked and signalled on the host it runs on, so a job recorded on another host is refused, naming the host to cancel it from, unless `other_host = "abandon"`: then nothing is signalled, and the run is recorded cancelled, with a reason saying so, and its progress file removed. Use it only once that host's session is gone and the job is confirmed not running there, for a render left running there is not stopped and would go on writing the job's output. A job whose process is already gone without recording its end is recorded failed, with that as its reason, once any of its render still running has been stopped. A job that has finished, failed or never run is left as it is. A submit or cancel in progress holds `<job>.run.lock`, as `submit_job` describes, and a cancel is refused while it is held. Not available on Windows.

Value

Invisibly, the job's status afterwards, as `job_status` returns it.

Examples

```
job <- file.path(tempdir(), "example.qmd")
writeLines("---\n\ttitle: example\n---", job)
if (.Platform$OS.type != "windows") cancel_job(job)
```

guard_long_fit	<i>Stop a long fit from starting in an interactive session</i>
----------------	--

Description

A template calls this just before an expensive `cache_fit()`. In an interactive session (the console, or running chunks one by one) with no cached result yet, it stops and says to run the job with `submit_job` instead, because a fit that runs for hours dies with the session. A render, a cached result, or `options(hvtiRtemplates.allow_interactive_fit = TRUE)` lets it pass.

Usage

```
guard_long_fit(name, dir, why, refit = FALSE)
```

Arguments

name	Character(1). The <code>cache_fit()</code> name; the cache file is <code><dir>/<name>.rds</code> .
dir	Character(1). The <code>cache_fit()</code> directory.
why	Character(1). Why the fit is long, shown in the error.
refit	Logical(1). TRUE when the job will recompute the fit even though a cached file exists (the template's <code>REFIT</code>); the cached file then does not make the call safe, so it is refused too.

Value

`invisible(FALSE)` when the fit may proceed; otherwise an error.

Examples

```
guard_long_fit("example-fit", dir = tempdir(), why = "it takes hours")
```

job_status	<i>Report a background job's state</i>
------------	--

Description

Reads the status `submit_job` keeps beside a job, `<job>.run.yml`. A job whose process was killed never records its end, so a recorded running is checked against the process itself. A render also keeps `<job>.run.progress`, read here; a render started without `submit_job`, after the last submitted run or with none, is reported from that file alone. A progress file older than the run the status file records is ignored.

Usage

```
job_status(path, quiet = FALSE)
```

Arguments

path	Character(1). The job's .qmd file.
quiet	Logical(1). FALSE prints a one-line summary.

Details

A long step writes a line to the render log at most every `getOption("hvtiRtemplates.progress_every", 10)` minutes.

Value

Invisibly, a list: state is one of "none", "submitted", "running", "finished", "failed", "vanished" or "cancelled" (stopped by `cancel_job`); the other elements are what the status file recorded (pid, created, host, started, ended, exit_code, elapsed_s, r_version, reason), and log, the log file's path. progress, the render's progress record when there is one, with eta_s, the estimated seconds left in the current stage, and stale, TRUE when it has not changed for three reporting intervals and at least 30 minutes.

Examples

```
job <- file.path(tempdir(), "example.qmd")
writeLines("---\ntitle: example\n---", job)
job_status(job)
```

life_table	<i>Kaplan-Meier life tables, as the ac job reports them</i>
------------	---

Description

The R counterpart of the SAS %KAPLAN macro, as the `ac` job calls it. `life_table()` returns the product-limit life table, overall or one per stratum. `life_table_at()` reads survival and the number at risk at reporting horizons, `life_table_tests()` compares strata, and `life_table_keep()` names the rows all three analyze, so a figure can be drawn over exactly the same patients.

Usage

```
life_table(  
  data,  
  time,  
  event,  
  strata = NULL,  
  conf_level = 0.68268948,  
  event_only = TRUE  
)  
  
life_table_at(  
  data,
```

```

    time,
    event,
    times,
    labels = NULL,
    strata = NULL,
    conf_level = 0.68268948
)

life_table_tests(data, time, event, strata)

life_table_keep(data, time, event, strata = NULL)

```

Arguments

<code>data</code>	A data frame, one row per patient.
<code>time</code>	Name of the follow-up time column.
<code>event</code>	Name of the event column, 0/1 or logical.
<code>strata</code>	Name of one stratification column, or NULL for the whole cohort.
<code>conf_level</code>	Confidence level of the limits. Default 0.68268948.
<code>event_only</code>	TRUE returns a row per event time only (SAS PEVENT=1); FALSE adds a row per censoring time (PEVENT=0).
<code>times</code>	Reporting horizons, in the unit of time.
<code>labels</code>	Labels for times. NULL labels each horizon with its value.

Details

Rows analyzed. A row is analyzed when its time and event are both recorded. With `strata`, a row whose stratum value is missing, or is a mean-imputed level (`hvtiRutilities::imputed_levels()`), is dropped too, as %KAPLAN drops "invalid strata values". The dropped rows are not silent: the result carries them in its "dropped" attribute, one row per reason, and the `ac` job prints that table.

Confidence level. The default `conf_level` is %KAPLAN's CLEVEL default, 0.68268948: one standard error, not 95%.

Reporting horizons. Survival at a horizon is the last estimate at or before it, and 1 before the first event. Beyond the last observation the estimate is not defined and is NA, unless survival has already reached zero. The number at risk is the number of patients whose time is greater than the horizon: the at-risk count on the first life-table row AFTER the horizon, when every censored time has a row (PEVENT=0). The estimate comes from the row before, the count from the row after.

Tests. `life_table_tests()` reports the three tests SAS PROC LIFETEST prints for strata: the log-rank, the Wilcoxon (Gehan's weights, the number at risk) and $-2\text{Log}(\text{LR})$, the likelihood ratio test of an exponential model. All three use every analyzed row to the end of follow-up. To compare curves only up to a horizon, censor the data there first, in the job.

Value

`life_table()`: a data frame with the columns of `TemporalHazard::hzt_kaplan()` (`time`, `n_risk`, `n_event`, `n_censor`, `survival`, `std_err`, `cl_lower`, `cl_upper`, `cumhaz`, `hazard`, `density`, `life`),

led by stratum when strata is given. Its attribute "counts" has a row per stratum with n, n_events, n_censored and max_fup, the last observed time; "dropped" is described under Details, or NULL.

life_table_at(): a data frame with a row per stratum and horizon: stratum when strata is given, then horizon, time, survival, cl_lower, cl_upper and n_risk, with the "dropped" attribute.

life_table_tests(): a data frame with a row per test: test, chisq, df and p_value, with the "dropped" attribute. A stratum with nobody at risk at any event time adds nothing to the log-rank or Wilcoxon test and is left out of its degrees of freedom.

life_table_keep(): a logical vector, TRUE for each analyzed row of data.

Examples

```
if (requireNamespace("TemporalHazard", quietly = TRUE)) {
  d <- data.frame(
    years = c(0.2, 0.5, 0.9, 1.3, 2.0, 2.4, 3.1, 4.0, 5.5, 6.0),
    dead = c(1, 0, 1, 1, 0, 1, 0, 1, 0, 0),
    group = c("a", "b", "a", "b", "a", "b", "a", "b", "a", NA)
  )
  life_table(d, "years", "dead")
  life_table_at(d, "years", "dead", times = c(1, 2, 5), strata = "group")
  life_table_tests(d, "years", "dead", "group")
}
```

migrate_job

Migrate a legacy job into a supported analysis template

Description

Reads a legacy source job and copies its deterministic choices into a supported template. Writes the migrated job and an evidence report beside it. Review the remaining EDIT: markers before rendering the job.

Usage

```
migrate_job(
  source,
  subject,
  type,
  prefix = NULL,
  qualifier = NULL,
  lst = NULL,
  log = NULL,
  reference = NULL,
  dir = NULL
)
```

Arguments

source	Path to one legacy source job, relative to the working directory or absolute.
subject	Grouping topic for the new job's filename. It is a statistical endpoint only when the job analyses one; an endpoint-free job may use a topic such as "cohort".
type	Analysis-type field for the new job's filename.
prefix	Template prefix, such as "dc", or a template's full name, such as "dc-tables", which carries its qualifier and leaves qualifier NULL. Read from the SAS filename when NULL. When given without qualifier, the qualifier is still read from the filename's second field if that field names one of this prefix's templates.
qualifier	Template qualifier, such as "tables". Read from the SAS filename when NULL. When given without prefix, the prefix is read from the filename and this qualifier is used. Filename fields must match <code>[A-Za-z0-9_]+</code> .
lst	Optional path to a SAS listing, relative to the working directory or absolute. Defaults to the same-named file beside source when present.
log	Optional path to a SAS log, relative to the working directory or absolute. Defaults to the same-named file beside source when present.
reference	Optional vector of paths to output references, such as RTF or DOCX files, relative to the working directory or absolute. They record comparison targets, not analysis choices.
dir	Any directory in the study, used only to locate the study root. Defaults to the directory of source.

Details

A converter is available for dc-tables, dc-gfup, dc-trends, and dc-eda, each interpreting its own source choices; choices the interpreter does not recognize remain for review. Legacy EDA reports migrate into dc-eda with `SECTIONS <- c("continuous", "percent", "count")`, the pages they drew. Naming a deprecated template warns and writes the job from its replacement. Naming a template by its name before a rename, such as dp-trends, warns and writes the job from the renamed template, dc-trends; a legacy source whose own file name carries the old name, such as dp.trends.sas, migrates into the renamed template without a warning, since that name is not the caller's to change. A template with no converter yet still migrates: it is scaffolded with every `EDIT:` marker kept, the evidence travels with it, and the report says the migration adapter is not yet available.

Relative source, lst, log and reference paths resolve against the working directory, as in any R function; dir only locates the study root. Each file must exist, be readable and lie beneath that root, and symbolic links are checked after resolution. The evidence files are never modified. The report records whether the listing and log were supplied, found beside the source, or not found.

The job uses the filename and study folder selected by [add_job](#). Its report replaces the .qmd extension with `-migration.md` and records study-relative evidence paths, SHA-256 checksums, the package version, translated choices, unresolved choices, and ignored material. Source text quoted in the report is masked, whichever converter ran: string literal contents (R raw strings included), SAS and R comment bodies, `%let` values (whole, even when a macro-quoting function such as `%str()` holds a semicolon), `%put` text, unquoted title and footnote text, and digit runs of five or more are replaced by placeholders such as `"[string]"` or `[text]`, and Quarto or R Markdown prose and

YAML are withheld. Statement keywords, variable names and operators remain. Absolute paths in any remaining text are redacted. The job is not masked. SAS log errors leave a blocking EDIT: marker in the generated job. Listings, RTF files and log messages may contain patient observations. Their text is withheld; locations remain available for local review. Logs retain severity, error codes and recognized aggregate counts only.

Both outputs are prepared as temporary files beside their targets before placement. Migration refuses to overwrite an existing target, checked for each output immediately before it is placed. Outputs are placed by hard link, which cannot replace an existing file; where the filesystem refuses hard links, the prepared file is renamed into place after a fresh check, so a concurrent writer landing between that check and the rename can still be replaced. If placement fails, only the outputs this call placed are removed, along with its temporary files.

Value

The migrated job path, invisibly. The report is written beside it.

See Also

[add_job](#), [template_list](#)

Examples

```
root <- file.path(tempdir(), "migrate-job-example")
invisible(hvtiRutilities::study_setup(root, "Example", 1L, renv = FALSE))
# A legacy follow-up job; the template is read from its name, dc.gfup.
sas <- file.path(hvtiRutilities::study_dir("descriptive", root), "dc.gfup.sas")
writeLines(c("data followup;", " set built;", "run;",
             "proc means data=followup; var iv_dead; by dead; run;"), sas)
job <- migrate_job(sas, subject = "cohort", type = "eda")
basename(job)

# The choices it translated into the new job ...
grep("^(EVENT|FOLLOWUP) <- ", readLines(job), value = TRUE)

# ... and the report beside it, saying what was translated and what is left
# to review.
report <- readLines(sub("[.]qmd$", "-migration.md", job))
headings <- grep("^## ", report)
first <- which(report[headings] == "## Translated")
# The Translated section and the one after it, Unresolved.
report[headings[first]:(headings[first + 2L] - 1L)]
unlink(root, recursive = TRUE)
```

Description

Finds the study root from `dir`, creates the job with [add_job](#) when it does not exist, and opens it in the editor. An existing job is opened as it stands, never overwritten.

Usage

```
open_job(prefix, subject, type, dir = NULL, qualifier = NULL)
```

Arguments

prefix	Job type: one of the prefixes reported by template_list , or a template's full name as reported in its name column, e.g. "dc.trends"; the dash spelling "dc-trends" is accepted too. A full name carries the qualifier, so qualifier must then be left NULL.
subject	Grouping topic for the job set, e.g. "death" or "cohort". A subject names a statistical endpoint only when the job analyses one. Your choice: there is no list of valid values, and every job of one analysis should share it (see Details). Must match <code>^[A-Za-z0-9_]+\$</code> : <code>.</code> separates the filename's fields, and <code>-</code> separated them in names before 2026-10, so neither may appear here.
type	The analysis type the job's set belongs to, e.g. "hz". Your choice, like subject, and shared the same way. Must match <code>^[A-Za-z0-9_]+\$</code> , for the same reason as subject.
dir	Character. Any directory inside the study. NULL, the default, starts from the working directory, and is an error outside a study.
qualifier	Job type within the prefix, e.g. "trends" for dp. Required only where a prefix carries more than one template; omitting it there is an error naming the choices, never a silent pick. Restricted to <code>[A-Za-z0-9_]+</code> , because <code>.</code> separates the filename's fields.

Details

The study root is the nearest directory at or above `dir` holding `_study.yml`, so this can be called from anywhere inside a study. Naming, prefix and qualifier rules are those of [add_job](#). A job scaffolded before 2026-10, named `<subject>-<type>-<prefix>[-<qualifier>].qmd`, is found and opened under that name. So is a job scaffolded under a template's name before a rename, such as `graphs/dp.trends.<subject>.<type>.qmd` for `dc.trends`, whichever of the two names the call uses. The editor is opened only in an interactive session.

Value

The path to the job file, invisibly.

See Also

[add_job](#), [render_job](#)

Examples

```
root <- file.path(tempdir(), "open-job-example")
suppressMessages(hvtiRutilities::study_setup(
  root, study = "Example", study_tracker_id = 1L, renv = FALSE
))
open_job(prefix = "ac", subject = "death", type = "eda", dir = root)
unlink(root, recursive = TRUE)
```

read_job_data	<i>Read a job's data, keep its rows, and record what was done</i>
---------------	---

Description

The shared data step of every analysis template. It reads a registered dataset (or an hvtiRdatabuild analysis set), resolves the patient identifier, drops the medical record number columns, keeps the rows where selects and checks that rows are unique on key.

Usage

```
read_job_data(
  cfg,
  dataset = "study",
  analysis_set = NULL,
  where = NULL,
  id = "ccfid",
  key = NULL,
  join = NULL,
  join_vars = NULL,
  reduce = NULL,
  join_key = NULL,
  one_row_per_patient = FALSE,
  imputation = NULL
)
```

Arguments

cfg	Study configuration, from study_config .
dataset	Name of a dataset registered in <code>_study.yml</code> ; "built" and "study" both name the study dataset, and the selection records "study".
analysis_set	Name of an analysis set written by <code>hvtiRdatabuild::write_analysis_set()</code> , or NULL to read dataset whole. Analysis sets derive from the study dataset only.
where	Rows to keep: NULL, one condition from <code>quote()</code> , or a list from <code>rlang::exprs()</code> , all of which must hold. Conditions follow <code>dplyr::filter()</code> : a row where a condition is NA is dropped. A value from outside the data, written <code>.env\$min_age</code> or as a name that is not a column, is fixed into the condition when the data are read, so the recorded condition rebuilds the same rows wherever it runs. Only a

plain name and `.env$x` are fixed in; `x$y` and `x@y` are kept as written, and a name for a function stays a name. Such a value must be a vector, NULL, a symbol or a call; a data frame, list, environment or S4 object stops, since it would be saved whole. A condition that mentions the `id` column or a column named MRN or eMRN (ignoring case), directly or as `.data$x` or `.data[["x"]]`, stops before any row is filtered, because each condition is saved, values included, in the job's output. Exclude those patients in the dataset build, or with an `hvtiRdatabuild` analysis set, instead. A value from outside the data is fixed in before this check, so `.data[[nm]]` is judged by the column `nm` names. A column reached any other way, as by `.data[[paste0(...)]]` or a string lookup such as `get()`, stops too, since it cannot be known. A function bound to such a lookup under another name is read as that lookup. Values are checked as well as names, after every name check: the check covers literal values (including a vector fixed in from outside) of five or more characters that equal a value of the identifier, MRN or eMRN columns in the data, and folds only arithmetic, `c()`, `paste/paste0` and numeric or character coercion of constants (such as `4730000000 + 1`) to check their result. So a wrapper function or a copy of the identifier under another name is caught when it is compared with such a value. The check never calls any other function: a value produced by any other call, or computed from a data column, as in `id2 / 2 != 2365000000.5`, is not checked. A threshold that happens to equal a patient's identifier is refused too, and the message says so; numbers are compared as whole numbers where they are whole, and NA never matches. Thresholds shorter than five characters, such as 1, 18 or 2015, are not checked, so a study keyed on shorter identifiers relies on the name checks.

<code>id</code>	The patient identifier column. When it is the default "ccfid" and absent, MRN and then eMRN are used.
<code>key</code>	Columns that make a row unique. NULL (the default) uses the key registered for dataset, and <code>id</code> , one row per patient, when none is registered. Add a visit time or date for repeated measures. A key that differs from the registered one is noted in the record. Templates pass their KEY, a study choice the analyst reviews, so the default serves a direct call. With <code>join</code> , the cohort must be one row per patient. A key of cohort columns is checked on the cohort, and the result is keyed as <code>reduce</code> says; a key that names a joined column, such as a visit time only the joined dataset carries, is checked on the joined rows and keys the result.
<code>join</code>	Name of one registered ancillary dataset (such as <code>echoes</code> or <code>labs</code>) to join to the cohort on <code>id</code> , or NULL. The cohort, dataset or <code>analysis_set</code> , decides the patients: joined records of other patients are dropped and counted. A dataset registered with a kind other than "ancillary" is refused.
<code>join_vars</code>	Cohort columns each joined row carries, the <code>id</code> always among them; NULL carries all of them. A column both datasets have stops, so list only the cohort columns the job needs.
<code>reduce</code>	NULL keeps one row per joined record, keyed on the joined dataset's key. <code>list(rule = "first", by = "echo_date")</code> keeps one row per cohort patient, keyed on <code>id</code> : the record with the smallest by ("first"), the largest ("last"), or the one nearest a cohort column (<code>rule = "nearest" with to = "dt_surg"</code>). A patient with no record keeps the row with the joined columns missing. <code>by</code> may name several columns, such as <code>c("echo_date", "echo_seq")</code> : each later one breaks a tie on

	those before it, in the same direction, and a missing tie-break value loses the tie. Records with no value in the first by column are left out and counted, and a patient whose chosen record still ties with another stops the read, since picking one would be a hidden choice.
join_key	Columns that make the joined dataset's rows unique, overriding its registered key. A join needs one or the other.
one_row_per_patient	TRUE for a job that models one row per patient, such as a hazard, logistic or random forest fit. Such a job stops on a join without reduce, before any data are read, since the long form would count every joined record as a patient, and on rows kept that repeat a patient, as a dataset of repeated records read whole would, after the rows are selected. FALSE, the default, accepts the long form, as descriptive jobs and jobs that model repeated measures need.
imputation	NULL, the default, or the name of the column that numbers the imputations of stacked multiple imputations, one row per patient per imputation. With one_row_per_patient = TRUE the rows are then counted within each imputation, so a patient who appears once in every imputation passes, and one who appears twice in the same imputation still stops the read. The column must be in the data read, and in key for the rows to be unique. It is matched ignoring case, as key is. It must hold no missing value and must not be the patient identifier. The record says how many imputations the rows hold. It is not inferred from key, since a visit column in key makes a long dataset, not an imputation.

Details

Direct identifiers are dropped unless one is the identifier, from a joined dataset too: columns named, ignoring case, ccfid, MRN, eMRN, a Social Security number (ssn, soc_sec, social_security, pt_ssn), and, held as text, a patient name (name, first_name, pt_name, surname) or contact details (address, street, phone, email). Any other column whose values are shaped like Social Security numbers is dropped too, and the record notes it by name; a column named as a date or time is not read for that shape. A KEY column is never dropped. The record lists every column dropped. An explicit id or key matches its column ignoring case, because hvtiRutilities::read_built() lowercases column names. Identifier, key and date values are not printed: the record holds counts, and a where condition that mentions a key column, uses .data or calls a string lookup such as get() is shown, in the record and in error messages, with its values replaced by <value>. So is a condition on id in a selection saved before such conditions were refused. Every setting is checked before the data are read, and join's registration too; only what needs the data, such as whether a column exists, is checked after. With join, where applies to the joined rows, so a condition may name a column from either dataset, though not a cohort column join_vars leaves out; the joined dataset's identifier values are refused in it as the cohort's are. With reduce, a condition that names a column of the joined dataset filters its records before one is chosen per patient, so rule = "last" with echo_type == "TTE" keeps each patient's last such echo, and a patient left with no record is counted as having none. The other conditions filter the reduced rows. The data table shows them in that order; the selection records them as written.

Value

A list:

- data, the selected rows;
- record, a data frame of step and value to print, whose "selection" attribute holds the settings used. Its "Source" row names the file read, the dated version for a dataset registered as one. When the dataset's source has been rebuilt since it was registered, the registered version is read and a final "Note" row says so, naming `hvtiRutilities::update_manifest()`, the call that registers the new file. A stale analysis set read in a draft gets a "Note" row the same way. With join, rows name the joined dataset and the rows read from it, count its records outside the cohort and the cohort patients with none, and name the reduction;
- provenance, the read's provenance record;
- provenance_join, the joined dataset's provenance record, naming the version read, or NULL without join;
- attrition, an analysis set's per-rule attrition table, or NULL for a dataset read whole.

The "selection" attribute is a list that a job saves in its hand-off, so a downstream job can rebuild the same rows:

- dataset, with "built" recorded as "study", and analysis_set, as given;
- where, the exact text of each condition, values included, used to rebuild the rows; it stays inside the study and is never printed;
- where_shown, the same conditions as a report may show them, with the values of any condition on a key column, or using `.data` or a string lookup, replaced;
- id and key, the resolved column names, the key being the joined result's when there is a join;
- imputation, the resolved name of the imputation column in the data's own spelling, present only when imputation was given, so a job can pass it on to a fit;
- join, join_vars, reduce and join_key, the join as read, with join_vars, join_key and reduce's columns spelled as the data spell them and reduce's fields in name order, so the same join written two ways records the same; and cohort_key, the cohort's own key. All five are present only with a join;
- rows and patients, the counts kept;
- key_hash, a SHA-256 hash of the kept key values, with a join the joined dataset's key values too, so a downstream job can tell that it rebuilt the same patients, and with reduce the same chosen records, and not only the same counts.

Examples

```
root <- file.path(tempdir(), "job-data-example")
dir.create(root)
hvtiRutilities::study_setup(root, "Example", 1L, adopt = TRUE, cleanup = FALSE, renv = FALSE)
d <- data.frame(ccfid = 1:4, age = c(15, 40, 55, 70))
utils::write.csv(d, file.path(hvtiRutilities::study_dir("datasets", root), "built.csv"),
  row.names = FALSE)
hvtiRutilities::register_data(root, "built.csv")
cfg <- hvtiRutilities::study_config(start = root)
job <- read_job_data(cfg, where = quote(age >= 18))
job$record
unlink(root, recursive = TRUE)
```

render_job	<i>Render a job</i>
------------	---------------------

Description

Renders a job from its own directory. By default the render is a draft: open EDIT: markers appear in the report's DRAFT banner. With `final = TRUE` an unfinished job stops instead, as the render of an accepted result should.

Usage

```
render_job(path, final = FALSE, quiet = FALSE)
```

Arguments

<code>path</code>	Character. Path to a job .qmd file.
<code>final</code>	Logical. TRUE for the accepted result.
<code>quiet</code>	Logical. Passed to <code>quarto::quarto_render()</code> .

Details

`final = TRUE` sets `HVTI_TEMPLATE_STRICT` to 1 for this render only and restores its previous value afterwards, including after an error. The job's own edit guard decides whether it is finished; this function does not search for markers itself, so it cannot disagree with a render started from the editor or from Quarto.

Quarto starts its own R to run the job's chunks, and by default it takes whichever R comes first on the `PATH`, which on a server with several versions installed need not be the R that called this function. A study whose `renv` library belongs to the calling R then fails to find its packages. So the render sets `QUARTO_R` to this R's `bin` directory, `R.home("bin")`, and restores it afterwards. A non-empty `QUARTO_R` that is already set is left alone, and Quarto honors it with a Quarto CLI of 1.8.13 or later. With an older CLI the quarto package itself sets `QUARTO_R` to the calling R for every render, so a value set by the user does not reach Quarto there.

Jobs scaffolded by [add_job](#) capture their data provenance while executing and embed it in the completed HTML. The same project hooks used by the Render button and bare Quarto commands then publish a same-stem `.provenance.json` sidecar beside the actual output. A failed execution that leaves the prior HTML untouched also leaves its sidecar in place. If publication fails after an output changes, the hooks expose a prior sidecar only when its recorded output hash still matches. Otherwise they withhold the sidecar and retain its recovery backup with a warning.

Render a job rather than running its chunks in the console: a render starts in a clean session, so the report comes from this job alone. A console run, chunk by chunk or with Run All, still gets through every chunk but the last. That chunk records provenance, which needs a render, so a console run stops there with an error of class `hvtiRtemplates_not_rendered` that says to render the job. Everything computed above it is still in the session.

One job renders at a time in a study. A render started while another is still running in the same study stops with a message naming the running Quarto process; render again once it finishes. A lock left by a render that crashed is taken over automatically.

Value

path, invisibly.

See Also

[open_job](#)

Examples

```
if (requireNamespace("quarto", quietly = TRUE) && quarto::quarto_available()) {
  root <- file.path(tempdir(), "render-job-example")
  invisible(hvtiRutilities::study_setup(root, "Example", 1L, renv = FALSE))
  d <- data.frame(ccfid = 1:6, dead = c(0, 1, 0, 1, 0, 0),
                 iv_dead = c(1.2, 0.4, 3.1, 2.2, 0.8, 4.5))
  utils::write.csv(d, file.path(hvtiRutilities::study_dir("datasets", root), "built.csv"),
                  row.names = FALSE)
  invisible(hvtiRutilities::register_data(root, "built.csv"))
  job <- add_job("dc.gfup", subject = "cohort", type = "eda", dir = root)

  # A draft: the new job still holds EDIT: markers, so the report carries a
  # DRAFT banner listing them, and renders all the same.
  render_job(job, quiet = TRUE)
  print(file.exists(sub("[.]qmd$", ".html", job)))

  # A final render is the accepted result, so it stops while any of the
  # job's open markers remains. This many:
  print(sum(grepl("EDIT:", readLines(job), fixed = TRUE)))
  try(render_job(job, final = TRUE, quiet = TRUE))
  unlink(root, recursive = TRUE)
}
```

stop_here

Render a job only down to this point

Description

Called in a chunk of a job, `stop_here()` ends the render there: the report keeps everything above it and leaves out everything below. It is how a part-built job is rendered, in place of commenting its unfinished sections out. To leave out one chunk in the middle instead, give that chunk a skip option with the reason: `#| skip: "waiting on the corrected coding"`.

Usage

```
stop_here(envir = parent.frame())
```

Arguments

envir	The environment the job's chunks run in, where its data provenance, SUBJECT and TYPE are found. The default is right inside a chunk.
-------	--

Details

Both are allowed in a draft and refused in a final render, by the same rule as an unresolved EDIT: marker. A draft lists each one, with its line and reason, in a callout at the top of the report, so a partial report cannot pass for a whole one. `render_job` with `final = TRUE`, or any render with `HVTI_TEMPLATE_STRICT` set, stops while any remains. Both are found by reading the job's source, so they show in a diff and in review, as commented-out code does not. A `stop_here()` the source scan cannot see, as inside an `if ()` or called with an argument, refuses a final render itself when it runs. In a draft it is recorded in the report's provenance as a stop, the run facts there and in the "About this run" table report a draft that stopped early (the box under the title was printed before the stop ran, and still reads as it did at the start), and a notice above the table says the run ended as a draft and why.

A skip must give a reason as a non-empty quoted string; `skip: true` is an error. A skipped chunk is neither run nor shown, so a later chunk that needs what it would have made fails: use `stop_here()` above both instead.

Outside a render, run line by line or with Run All, `stop_here()` does nothing, so the chunks below it still run interactively.

A job's provenance is embedded by its last chunk, which a stopped render never reaches, so `stop_here()` embeds it instead: the data the job read, its subject and type, and the partial-render points. Call it on its own line at the end of a chunk, so the chunk shows what it returns.

Value

Outside a render, `NULL`, invisibly. Inside one, the job's provenance block, for the chunk to print as the last output of the report.

See Also

[render_job](#)

Examples

```
# Outside a render it does nothing.
stop_here()
```

submit_job

Render a job in the background, detached from the session

Description

Starts `render_job` on the job in a separate process that outlives the R session, the way `nohup` from a terminal does, and returns at once. Use it for a job whose fit runs for hours: a Workbench session that times out takes a console run with it.

Usage

```
submit_job(path, final = FALSE)
```

Arguments

path	Character(1). The job's .qmd file.
final	Logical(1). Passed to render_job .

Details

The process runs the Rscript of the R that called `submit_job()`, from the study root, so the study's `renv` and `.Renv` apply. It writes `<job>.run.log` (everything the render prints) and `<job>.run.yml` (its state) beside the job; read the state with [job_status](#). A job already running is not started twice: a `<job>.run.lock` directory is taken while a submit is in progress, and a submit that crashed can leave it behind, in which case delete it to try again. Stop a submitted job with [cancel_job](#). Not available on Windows, which cannot detach a process this way.

Value

Invisibly, the job's status, as [job_status](#) returns it.

Examples

```
# Not run by check: it would start a real render.
## Not run:
submit_job("analyses/nb.boostmtree.lvef.boost.qmd")

## End(Not run)
```

template_catalog

Catalog of analysis templates owed by this package

Description

The catalog records one row per job type, including templates still queued or blocked on functions in other packages. [template_list\(\)](#) reports the files already shipped; this catalog records the full work plan.

Usage

```
template_catalog()
```

Value

A data frame. `uses`, `upstream`, `downstream`, and `workflows` are list columns of character vectors. Unmeasured counts are `NA_integer_`. `description` is a one-sentence summary, given for every template on disk and `NA` for most queued ones. `deprecated_by` names the template replacing a deprecated one, such as "dc-gfup", and `deprecation_note` says how to move to it; both are `NA` for a supported template. A deprecated template still ships, and [add_job](#) warns when it is used.

Examples

```
catalog <- template_catalog()
table(catalog$status)

# Templates that still scaffold but name a replacement.
catalog[!is.na(catalog$deprecated_by), c("prefix", "qualifier", "deprecated_by")]

# Queued templates waiting on work in another package.
queued <- catalog[catalog$status == "queued" & !is.na(catalog$blocked_on), ]
queued[, c("prefix", "qualifier", "blocked_on")]
```

template_list	<i>List the supported R job templates</i>
---------------	---

Description

These templates are supported: they render, they are tested, and they are the intended starting point for a new analysis job.

Usage

```
template_list()

## S3 method for class 'hvti_template_list'
print(x, ...)
```

Arguments

x	A data frame returned by <code>template_list()</code> .
...	Passed to the data frame print method.

Details

A template is named `<prefix>.qmd`, or `<prefix>-<qualifier>.qmd` where one prefix carries several job types, and lives in a numbered directory named for the taxonomy folder it scaffolds into, so folder is read from the tree rather than looked up. A qualified template is shown, and selected, as `<prefix>.<qualifier>`, e.g. `dc.trends`, the spelling a job's name uses; the file keeps its dash, and `"dc-trends"` is still accepted as input. The directory's leading digits order the folders and are stripped from folder. The placement test requires the job catalog and skips when it is absent. Its internal lookup helper uses the catalog's (prefix, qualifier) row, falling back to [hvti_taxonomy](#) when the catalog or matching row is absent. A separate test checks that every template directory names a taxonomy folder, including when the catalog is absent.

Value

A data frame, of class `hvti_template_list`, with columns `name`, `prefix`, `qualifier`, `folder`, `call` and `file`. `call` is the [add_job](#) call that scaffolds the template, with only the arguments it requires and runnable as printed, e.g. `add_job("dc.gfup", subject = "cohort", type = "eda")`. The subject and type shown are the template's own defaults; change them to name the job. `folder` is the taxonomy name with the directory's ordering digits stripped, so `20_distributions` reports as `distributions`. `qualifier` is NA for a prefix carrying a single template. Printed whole, it shows only name, prefix, qualifier and folder, since `call` and `file` would wrap every row; both are still in the data, and a selection of columns prints as selected.

Examples

```
tl <- template_list()
# Printed whole: name, prefix, qualifier and folder.
tl

# Every template, where its job is written, and the call that scaffolds it.
tl[, c("name", "folder", "call")]

# Find a template by its qualifier, then copy its call: the follow-up jobs.
subset(tl, qualifier == "gfup", c(name, folder, call))
```

template_path	<i>Path to a supported template</i>
---------------	-------------------------------------

Description

Path to a supported template

Usage

```
template_path(prefix, qualifier = NULL)
```

Arguments

prefix	Analysis prefix, e.g. "ac", or a template's full name, e.g. "dc.trends" (or "dc-trends"), which carries its qualifier and leaves <code>qualifier</code> NULL. See template_list .
qualifier	Job type within the prefix, e.g. "trends" for dc. Required only where a prefix carries more than one template; omitting it there is an error naming the choices, never a silent pick.

Value

The full path, as `character(1)`. A template the catalog marks deprecated still resolves, with a warning naming its replacement. So does a template's name before a rename, such as `dp.trends`, now `dc.trends`: it resolves to the renamed template, with a warning naming the new name, until the old name is removed.

Examples

```
template_path("ac")

# A qualified template, by its full name or as prefix plus qualifier.
template_path("dc.gfup")
template_path("dc", qualifier = "gfup")

# A prefix carrying several templates is never resolved by guessing: this
# is an error that lists the choices.
try(template_path("dc"))
```

upgrade_job

Rewrite a job from the current version of its template

Description

Upgrades **one job**: the file you name in path. It does not look for other jobs, so to upgrade several, call it once for each.

A job is a copy of its template as it stood on the day [add_job](#) wrote it. `upgrade_job()` writes the job again from the template as it stands now and carries the study's choices across, so a job keeps up with its template without re-entering every choice by hand.

Usage

```
upgrade_job(path, dir = NULL)
```

Arguments

path	The one job to upgrade: its .qmd file, relative to the working directory or absolute, such as "10_descriptive/dc.general.cohort.eda.qmd".
dir	Any directory in the study, used only to locate the study root. Defaults to the directory of path, which is right for any job inside its study.

Details

The template is read from the job's file name, in either spelling [add_job](#) has used. A study's choices are the top-level assignments to an upper-case name, such as `DATASET <- "built"`, in the job's `edit-*` chunks. Each choice the current template also has replaces the template's assignment, its source text unchanged, multi-line values included. `SUBJECT` and `TYPE` are not carried: the file name owns them. Where the job had resolved a choice's `EDIT:` marker, the upgraded job drops the template's marker for it too.

Nothing is dropped silently. A choice the current template no longer has is written, commented out, at the end of the template's `edit-study-choices` chunk under a new `EDIT:` marker, so the job renders as a draft until someone decides. Prose and code outside the choices come from the current template, because without the template the job was copied from an analyst's edit cannot be told from the old template's text. The report lists every chunk that may hold such an edit, and every comment line above a carried choice that the current template's comment lacks.

The one exception is the job's description. Where the template opens with an `EDIT:` comment asking for it, the text the old job has in that place is carried instead, once the study has deleted the comment's marker. While the marker stands, the report names those lines for review.

The old job is kept beside the new one as `<file>.pre-upgrade`, a name nothing renders. The upgraded job takes the name `add_job` would give it today, so a job named before 2026-10 moves to the `-separated` spelling. The report is written beside the job, with the `.qmd` extension replaced by `-upgrade.md`. It records the template, the package version, the old job's checksum, and the choices carried, not placed and new, with the chunks to review. It names choices and lines but never quotes their values, since a value can identify a patient. The upgrade refuses to overwrite any existing file.

Value

The upgraded job's path, invisibly. The report is written beside it.

Upgrading a job

Name the job's `.qmd` file. From the study root, a cohort EDA job is

```
upgrade_job("10_descriptive/dc.general.cohort.eda.qmd")
```

and several jobs are upgraded one call each:

```
for (job in c("10_descriptive/dc.general.cohort.eda.qmd",
              "10_descriptive/dc.general.cohort.gfup.qmd")) {
  upgrade_job(job)
}
```

You never name the template: the job's file name says which one it is. The job must sit inside its study; leave `dir` out unless you want to name a different directory of the same study.

Afterwards, render the job and read its report, `<job>-upgrade.md`, for the chunks to check. The old job is kept as `<job>.pre-upgrade`. Before upgrading the same job again, move both of those files aside: `upgrade_job()` never overwrites a file, so a second upgrade stops while either one is there.

See Also

[add_job](#), [migrate_job](#)

Examples

```
root <- file.path(tempdir(), "upgrade-job-example")
invisible(hvтиRutilities::study_setup(root, "Example", 1L, renv = FALSE))
job <- add_job("dc", "cohort", "eda", dir = root, qualifier = "general")
# A study choice, made in the job.
lines <- readLines(job)
lines <- sub('^DATASET <- "built"', 'DATASET <- "built_r"', lines)
writeLines(lines, job)
upgraded <- upgrade_job(job)
grep("^DATASET <- ", readLines(upgraded), value = TRUE)
unlink(root, recursive = TRUE)
```

Index

`add_job`, [3](#), [12–14](#), [19](#), [22](#), [24–26](#)

`build_cohort`, [5](#)

`cancel_job`, [6](#), [9](#), [22](#)

`guard_long_fit`, [8](#)

`hvti_taxonomy`, [23](#)

`job_status`, [6](#), [7](#), [8](#), [22](#)

`life_table`, [9](#)

`life_table_at` (`life_table`), [9](#)

`life_table_keep` (`life_table`), [9](#)

`life_table_tests` (`life_table`), [9](#)

`migrate_job`, [11](#), [26](#)

`open_job`, [4](#), [13](#), [20](#)

`print.hvти_template_list`
 (`template_list`), [23](#)

`read_job_data`, [15](#)

`render_job`, [14](#), [19](#), [21](#), [22](#)

`stop_here`, [20](#)

`study_config`, [15](#)

`submit_job`, [6–8](#), [21](#)

`template_catalog`, [4](#), [22](#)

`template_list`, [3–5](#), [13](#), [14](#), [22](#), [23](#), [24](#)

`template_path`, [5](#), [24](#)

`upgrade_job`, [25](#)