Package: submitr (via r-universe)

October 31, 2026

Title Scaffold and Submit Computational Jobs to CHTC Schedulers

Version 0.1.0.9000

Description Provides scaffolding tools to help researchers prepare and submit computational jobs to high-throughput computing (HTC) schedulers. Generates the files required to run containerized R analyses on 'HTCondor', including submit files and executable scripts, and wraps the system commands needed to stage files, submit jobs, monitor status, and retrieve results from a CHTC submit node. A submission state file stored alongside the project configuration tracks each submission across the workflow, so that input files can be uploaded and result files downloaded automatically, without constructing file lists by hand and without losing the job when the R session ends. Provides htc_start() for session management and htc_config() for connection details and SSH connection reuse guidance. Works naturally alongside 'containr' for container image management and 'toolero' for dataset splitting and project scaffolding.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

SystemRequirements SSH client, HTCondor (https://htcondor.org)

Roxygen list(markdown = TRUE)

Suggests containr, knitr, rmarkdown, spelling, testthat (>= 3.0.0), toolero, withr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Depends R (>= 4.1.0)

Imports cli, glue, readr, tibble, yaml

VignetteBuilder knitr

URL https://erwinlares.github.io/submitr/

BugReports https://github.com/erwinlares/submitr/issues

Repository https://erwinlares.r-universe.dev

Date/Publication 2026-10-01 15:23:33 UTC

RemoteUrl https://github.com/erwinlares/submitr

RemoteRef HEAD

RemoteSha fa835f8621bdea0c1825af0fb7fe99aff0be096c

Contents

htc_cancel 2
htc_check 4
htc_collect 6
htc_config 8
htc_download 11
htc_gen_executable 14
htc_gen_submit 18
htc_release 22
htc_ssh_setup 24
htc_start 25
htc_status 27
htc_submit 29
htc_upload 32

Index **35**

`htc_cancel` *Cancel HTCondor jobs*

Description

`htc_cancel()` connects to an HTC submit node via SSH and runs `condor_rm` to remove jobs from the queue – a mistaken submission (the wrong resource request, the wrong image, 500 jobs instead of 5) can be stopped from R rather than requiring a manual SSH session (S-G2).

Usage

```
htc_cancel(
  cluster_id = NULL,
  reason = NULL,
  config = NULL,
  dry_run = FALSE,
  verbose = FALSE,
  path = "."
)
```

Arguments

cluster_id	An integer, character string, or NULL. The cluster ID to remove, e.g. 6302860. When NULL (the default), resolves to the cluster ID recorded in the submission state by the most recent htc_submit() call. Unlike htc_status(), htc_cancel() never falls back to "all of my jobs": a bare condor_rm with no arguments removes every job you have queued on the submit node, not just the ones from this project, so the function errors instead of guessing when no cluster ID is available anywhere.
reason	A character string or NULL. An optional free-text reason recorded in the job's HTCondor log (via condor_rm's own -reason flag), useful when revisiting a log later to remember why a cluster was removed. Defaults to NULL.
config	A named list as returned by htc_config(). Must contain username and server. If NULL (the default), uses the session config set by htc_start(). If no session config is set, the function errors with instructions.
dry_run	Logical. If TRUE, prints the SSH command that would be executed without running it. Defaults to FALSE.
verbose	Logical. If TRUE, prints progress messages. Defaults to FALSE.
path	A character string. Directory holding the submission state (htc-manifest.yml), consulted only when cluster_id is NULL. Defaults to ".", matching the default used elsewhere in the family.

Value

The condor_rm output as a character vector, returned invisibly.

Workflow

```
cfg <- htc_config()
job <- htc_submit(config = cfg)

# Realized the resource request was wrong -- stop it before it runs
htc_cancel(cluster_id = job, config = cfg, reason = "wrong request_memory")
```

See Also

htc_release() to un-pause a held job instead of removing it, and htc_status() to see which jobs are running, idle, or held.

Examples

```
# Preview the SSH command without connecting to CHTC
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
htc_cancel(cluster_id = 6302860, config = cfg, dry_run = TRUE)


## Not run:
cfg <- htc_config()
htc_cancel(cluster_id = 6302860, config = cfg)
```

```
htc_cancel(cluster_id = 6302860, config = cfg, reason = "submitted by mistake")

## End(Not run)
```

`htc_check` *Preflight check for a submitr job before upload or submission*

Description

`htc_check()` verifies, locally and in seconds, several things that would otherwise only surface an hour later as a held or failed job on the cluster (S-G4): that every file the job depends on actually exists on this machine, that a `"multiple"`-mode job's subset files still match what `subdatasets.csv` expects, that the resource request looks plausible, and – best effort, when a container tool is available locally – that the container image looks reachable.

Usage

```
htc_check(
  container_image = NULL,
  input_files = NULL,
  data_files = NULL,
  resources = NULL,
  check_image = TRUE,
  verbose = TRUE,
  path = "."
)
```

Arguments

`container_image`
: A character string or `NULL`. The container image reference to check, e.g. `"registry.doit.wisc.edu/net` When `NULL` (the default), resolves to the `container_image` recorded in the submission state by `htc_gen_submit()`. When still `NULL`, image checks are skipped.

`input_files`
: A character vector or `NULL`. Local paths that are expected to exist before upload – normally the same value passed to `htc_gen_submit()`. When `NULL` (the default), resolves to `input_files` recorded in the submission state.

`data_files`
: A character vector or `NULL`. Local paths to data files expected to exist before a container image bakes them in – normally the same value passed to `htc_gen_executable()`. When `NULL` (the default), resolves to `data_files` recorded in the submission state.

`resources`
: A named list with `cpus`, `memory`, and `disk`, or `NULL`. When `NULL` (the default), resolves to the resolved resource values `htc_gen_submit()` recorded in the submission state (whichever preset, or `custom_resources`, was actually used).

check_image	Logical. If TRUE (the default), and podman or docker is on the local PATH, asks it whether container_image is pullable (see the section on the image check below). Set FALSE to skip that probe, which contacts the registry and can take up to 15 seconds – when working offline, for instance. The latest-tag check runs either way.
verbose	Logical. If TRUE (the default), prints a line for every check performed, not just the ones that found something. Set FALSE to only see problems.
path	A character string. Directory holding the submission state (htc-manifest.yml). Defaults to ".", matching the default used elsewhere in the family.

Details

Every argument can be resolved from the submission state, so the common case is htc_check() with no arguments at all, run right after htc_gen_submit() and htc_gen_executable() and before htc_upload().

Value

A tibble with one row per issue found, columns check, severity ("error" or "warning"), and message. Zero rows means nothing was found. Returned invisibly; call print() explicitly or assign it to inspect the detail behind the printed summary.

What counts as an error versus a warning

A missing input, data, or subset file is an "error": the job will not run without it, full stop. A resource request outside typical bounds, a container_image tagged latest, or an image that could not be confirmed pullable are "warning"s – each might be exactly what you intend, and none of them is something htc_check() can be certain about from the local machine alone. htc_upload()'s own check argument only blocks the upload on "error"s.

The image check is best effort

htc_check() only attempts to confirm container_image is pullable when podman or docker is found on the local PATH, via <tool> manifest inspect. A failure there is reported as a warning, not an error, and is not conclusive either way: it may mean the image genuinely does not exist, or simply that you are not logged in to the registry from this machine, or that the tool timed out. When neither tool is found, or check_image = FALSE, the probe is skipped and reported as such.

See Also

htc_upload()'s check argument, which runs this automatically and aborts before uploading if an "error"-level issue is found.

Examples

```
tmp <- withr::local_tempdir()
htc_gen_submit(
  container_image = "registry.doit.wisc.edu/netid/myimage:latest",
  resources       = "small",
  output          = tmp,
```

```
  path            = tmp
)
# check_image = FALSE skips asking podman or docker about the image,
# which would otherwise contact the registry.
htc_check(path = tmp, check_image = FALSE)
```

`htc_collect` *Unpack downloaded results and index them, one row per job*

Description

`htc_collect()` is the step after `htc_download()`. It extracts every job's results tarball into its own folder and returns the *job index*: a tibble with one row per job saying whether its results came back, where they landed, what files they hold, and where that job's HTCondor log files are. In `"multiple"` mode that is one row per subset, the HTC-side counterpart to the tibble `toolero::run_by_group()` returns for a local run.

Usage

```
htc_collect(
  tarballs = NULL,
  local_path = ".",
  extract_dir = NULL,
  overwrite = FALSE,
  path = "."
)
```

Arguments

`tarballs`	A named character vector or `NULL`. Local tarball paths to collect, with names giving each one's group id. When `NULL` (the default), resolved from the submission state: the same subset names and script stem `htc_download()` used to name the files it fetched into `local_path`.
`local_path`	A character string. The directory `htc_download()` saved tarballs into. Defaults to `"."`, matching `htc_download()`'s own default. Only consulted when `tarballs` is `NULL`.
`extract_dir`	A character string or `NULL`. Directory to extract into, one subdirectory per group id. When `NULL` (the default), defaults to `local_path`.
`overwrite`	Logical. If `TRUE`, deletes and re-extracts a group's subdirectory when it already exists. If `FALSE` (the default), any existing subdirectory is an error, raised before anything is extracted, so a second `htc_collect()` call never silently mixes stale and fresh results.
`path`	A character string. Directory holding the submission state (`htc-manifest.yml`). Defaults to `"."`, matching the default used elsewhere in the family. Consulted for the tarball names when `tarballs` is `NULL`, and in every case for the cluster ID, process numbers, container image, and results folder.

Details

It works for any analysis, whatever the script wrote into its results folder, and never opens the files themselves: their types are whatever each job chose to save, and no single reader is right for all of them. Read them with whatever wrote them, using `output_dir` and `files` (see the Workflow section).

A job whose tarball is missing, or will not extract, still gets a row, with `extracted = FALSE`, and the collection carries on with the other jobs. A job whose R script fails still sends its tarball back (see `htc_gen_executable()`), so a missing tarball means the job stopped before it got that far: the container did not start, a file it needed was not transferred, or the job was removed. Its `.err` and `.log` files, in the `err` and `log` columns when they were downloaded, are the place to look.

Value

The job index: a tibble with one row per job and these columns.

- `group_id` – the subset stem in `"multiple"` mode (`"adelie"` for `adelie.csv`), the script stem in `"single"` mode, or the name given in `tarballs`.
- `proc_id` – the job's HTCondor process number, from its position in the submission state; `NA` when it cannot be matched.
- `cluster_id` – from the submission state; `NA` when absent.
- `extracted` – whether the tarball was found and extracted.
- `n_files` – how many files the results folder holds; `NA` when not extracted.
- `has_record` – whether the results folder holds an output record (`project-manifest.json`, written by `toolero::generate_manifest()`). The file is only checked for, never read. `NA` when not extracted.
- `output_dir` – where the job's results folder landed; `NA` when not extracted.
- `files` – a list column of paths relative to `output_dir`.
- `log`, `err`, `out` – local paths to the job's three HTCondor files, found beside its tarball; `NA` when not there.
- `container_image` – from the submission state; `NA` when absent.

Workflow

```
htc_download(local_path = "downloads/")
index <- htc_collect(local_path = "downloads/")

# Which jobs came back?
index[, c("group_id", "extracted", "n_files")]

# Why did a job fail? Its .err file usually says.
readLines(index$err[!index$extracted][1])

# Full paths to every file each job produced
paths <- Map(file.path, index$output_dir, index$files)
models <- lapply(unlist(paths[index$extracted]), readRDS)
```

Examples

```
# Build a single-job tarball by hand to show what htc_collect() does with
# one -- normally this tarball would have come from htc_download() after
# a real job finished.
job_dir <- tempfile("job")
dir.create(file.path(job_dir, "output"), recursive = TRUE)
saveRDS(mtcars, file.path(job_dir, "output", "mtcars.rds"))

local_path <- tempfile("downloads")
dir.create(local_path)
tarball <- file.path(local_path, "analysis-results.tar.gz")
old <- setwd(job_dir)
utils::tar(tarball, "output", compression = "gzip", tar = "internal")
setwd(old)

index <- htc_collect(
  tarballs    = c(analysis = tarball),
  extract_dir = tempfile("collected")
)
index
index$files[[1]]
```

---

htc_config *Configure a connection to an HTC submit server*

Description

`htc_config()` creates or reads an `htc.cfg` file that stores the connection details needed by `htc_upload()`, `htc_submit()`, `htc_status()`, and `htc_download()`. On first use it prompts interactively for your username and server address, writes `htc.cfg` to `path`, and adds it to `.gitignore`. Subsequent calls read the existing file.

Usage

```
htc_config(
  username = NULL,
  server = NULL,
  path = ".",
  overwrite = FALSE,
  project_config = NULL,
  check_server = getOption("submitr.check_server", default = TRUE)
)
```

Arguments

`username`	A character string. Your HTC username (NetID), e.g. `"erwin.lares"`. If `NULL` and no `htc.cfg` exists, the function prompts interactively.

server — A character string. The HTC submit server hostname. Defaults to "ap2002.chtc.wisc.edu". If NULL and no htc.cfg exists, the function prompts interactively.

path — A character string. Directory where htc.cfg will be read from or written to. Defaults to "." (current working directory).

overwrite — Logical. If TRUE, recreates htc.cfg even if one already exists. Defaults to FALSE.

project_config — A character string or NULL (the default). Path to a _toolero.yml file, the resolved project configuration toolero::init_project() writes to a project's root. When supplied, its folders and conventions sections are parsed once and folded into the returned list under project, so config$project$folders and config$project$conventions become available alongside the connection details. This is the same file, in the same schema, that containr::generate_dockerfile() reads through its own config argument. Never written into htc.cfg: project layout and SSH connection details are recorded separately, and htc.cfg on disk is unaffected by whether you pass project_config.

check_server — Logical. If TRUE, opens a short SSH connection to server to report whether it is reachable before you rely on the config. Defaults to the submitr.check_server option, which is itself TRUE unless you set it otherwise. Set to FALSE in scripts, test suites, and anywhere else the probe has no audience – reading a config file then costs nothing and touches no network.

Value

A named list with elements username and server, plus a project element when project_config is supplied, returned invisibly.

Options

Two options adjust how much htc_config() does on your behalf. Both default to TRUE, and both are most useful set once for a whole session or test suite rather than per call.

submitr.verbose controls the progress messages ("Reading HTC config from ...", "Checking connectivity to ..."). Setting it to FALSE leaves warnings and errors intact.

submitr.check_server controls the reachability probe described under check_server above. The argument takes precedence when supplied, so the option sets the default and a call can still override it.

SSH connection reuse

Each call to htc_upload(), htc_submit(), htc_status(), or htc_download() opens a new SSH connection to the submit server, which triggers a Duo MFA prompt each time. You can avoid this by configuring SSH connection reuse (ControlMaster) in your ~/.ssh/config file. Add the following block:

```
Host *.chtc.wisc.edu
  ControlMaster auto
  ControlPersist 2h
  ControlPath ~/.ssh/connections/%r@%h:%p
```

Then create the connections directory:

```
mkdir -p ~/.ssh/connections
```

After this, only the first connection in a two-hour window will require Duo authentication. Full documentation: https://chtc.cs.wisc.edu/uw-research-computing/configure-ssh

Project configuration

`project_config` is how `submitr` learns the layout a `toolero` project already settled on, instead of retyping it. `toolero::init_project()` writes `_toolero.yml` to a project's root once it has resolved the folder set, and that file records two things: `folders`, the full list of folders the project uses, and `conventions`, the names the family resolves rather than assumes (`output_dir`, `script_dir`, `split_dir`).

Passing `project_config = "_toolero.yml"` reads that file once and returns it under `config$project`:

```
cfg <- htc_config(project_config = "_toolero.yml")
cfg$project$conventions$output_dir
#> [1] "output"
```

As of this release (S-G5), passing the returned config's `project` element on to `htc_gen_executable()` and `htc_gen_submit()` via their own `config` argument lets `results_folder` and `queue_from` default from `conventions$output_dir` and `conventions$split_dir`, the way `containr::generate_dockerfile()` already reads the same file for its own purposes:

```
cfg <- htc_config(project_config = "_toolero.yml")
htc_gen_executable(r_script = "R/analysis.R", config = cfg)
htc_gen_submit(mode = "multiple", config = cfg)
```

`r_script` itself is not defaulted from `conventions$script_dir`: the convention names a directory, not a file, and the script's own filename is project-specific information `_toolero.yml` has no way to record.

Security

`htc.cfg` contains your username and server address. Neither is sensitive on its own, but `htc_config()` adds `htc.cfg` to `.gitignore` on creation to avoid accidentally committing institutional account details to a public repository.

Examples

```
# Preview what htc_config() would return without writing any files
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
str(cfg)


## Not run:
# Interactive first-time setup
cfg <- htc_config()
```

```
# Non-interactive setup (for scripts)
cfg <- htc_config(
  username = "erwin.lares",
  server   = "ap2002.chtc.wisc.edu"
)

# Force recreation of htc.cfg
cfg <- htc_config(overwrite = TRUE)

# Read the config without probing the server, e.g. in a script or on CI
cfg <- htc_config(check_server = FALSE)

# Or turn the probe off for a whole session
options(submitr.check_server = FALSE)

# Fold in a toolero project's own folder layout and conventions
cfg <- htc_config(project_config = "_toolero.yml")
cfg$project$conventions$output_dir

# Use in other functions
htc_upload(files = c("job.sub", "job.sh"), config = cfg)

## End(Not run)
```

`htc_download` *Download files from an HTC submit node*

Description

`htc_download()` copies one or more files from a directory on an HTC submit node to a local directory via scp. It is the final step in the job submission workflow – called after `htc_status()` confirms all jobs have completed.

Usage

```
htc_download(
  files = NULL,
  cluster_id = NULL,
  remote_path = NULL,
  local_path = ".",
  config = NULL,
  dry_run = FALSE,
  verbose = FALSE,
  path = "."
)
```

Arguments

files	A character vector or NULL. One or more filenames or glob patterns to download from remote_path on the submit node. Examples: "results.tar.gz", c("job.log", "job.err"), "*.tar.gz". When NULL, the function uses cluster_id and the submission state to determine which files to download. Defaults to NULL.
cluster_id	A character string or NULL. The cluster ID returned by htc_submit(). When supplied without files, the function constructs the file list from the submission state. When NULL, falls back to the most recently submitted cluster ID stored in the submission state. Defaults to NULL.
remote_path	A character string or NULL. The directory on the submit node where the files are located. When NULL (the default), resolves to the remote_path recorded in the submission state by the preceding call to htc_submit(), falling back to "~/" if the submission state holds no value. Should match the remote_path used in htc_upload() and htc_submit().
local_path	A character string. The local directory where downloaded files will be saved. Defaults to "." (current working directory).
config	A named list as returned by htc_config(). Must contain username and server. If NULL (the default), uses the session config set by htc_start(). If no session config is set, the function errors with instructions.
dry_run	Logical. If TRUE, prints the scp command that would be executed without running it. Defaults to FALSE.
verbose	Logical. If TRUE, prints progress messages. Defaults to FALSE.
path	A character string. Directory holding the submission state (htc-manifest.yml). This is where the function looks for job metadata; it is not where downloaded files are written, which is local_path. Defaults to ".". If you passed a non-default output or path to htc_gen_submit() and htc_gen_executable(), pass that same directory here.

Details

When cluster_id is supplied without files, the function uses the submission state built up by htc_gen_submit(), htc_gen_executable(), and htc_submit() to determine which files to download. For single-mode jobs, this includes the results tarball and the log, error, and output files. For multiple-mode jobs, the function reads the subset names from the submission state and constructs per-job tarball names and per-process log file patterns.

Glob patterns such as "*.tar.gz" are supported when using the files argument and are evaluated on the remote server, not locally.

Value

Called for its side effects. Returns invisible(NULL).

Automatic file resolution

When files is NULL, the function resolves the file list from the submission state. The submission state is built automatically as you call htc_gen_submit(), htc_gen_executable(), and htc_submit() during the normal workflow. No extra steps are needed.

For a single-mode job:

- The results tarball (e.g. `"analysis-results.tar.gz"`)
- Log files: `"{cluster_id}-0-job.log"`, `".err"`, `".out"`

For a multiple-mode job:

- Per-subset tarballs (e.g. `"analysis-adelie-results.tar.gz"`)
- Log files for each process: `"{cluster_id}-{0,1,...}-job.log"`, etc.

Workflow

`htc_download()` is the final system-facing step in the submitr workflow. Call it after `htc_status()` confirms all jobs have completed.

```
# Automatic: uses the submission state to determine what to download
htc_start()
htc_gen_submit(...)
htc_gen_executable(...)
htc_upload(...)
job <- htc_submit("analysis.sub")
htc_status(cluster_id = job, watch = TRUE)
htc_download()
```

Glob patterns

When using `files` directly, glob patterns are passed to the remote shell for evaluation so they match files on the submit node, not on your local machine. The pattern is single-quoted in the `scp` command to prevent local shell expansion.

SSH connection reuse

Each call to `htc_download()` opens a new SSH connection. If you have not configured ControlMaster in your `~/.ssh/config`, this will trigger a Duo MFA prompt. Run `htc_config()` for setup guidance.

Examples

```
# Preview the scp command without connecting to CHTC
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
htc_download(files = "*.tar.gz", config = cfg, dry_run = TRUE)


## Not run:
# All remaining examples require a live CHTC connection

# Automatic download after a workflow
htc_start()
htc_gen_submit(...)
htc_gen_executable(...)
htc_upload(...)
```

```
job <- htc_submit("job.sub")
htc_status(cluster_id = job, watch = TRUE)
htc_download()

# Download by cluster ID
htc_download(cluster_id = "6590895")

# Download specific files using globs
htc_download(files = "*.tar.gz", local_path = "downloads/")

# Download logs only
htc_download(files = c("*.log", "*.err", "*.out"), local_path = "logs/")

## End(Not run)
```

`htc_gen_executable` *Generate an HTCondor executable shell script for an R job*

Description

`htc_gen_executable()` writes a ready-to-use bash script (`.sh`) that HTCondor runs inside the container for each job. The script changes to HTCondor's writable scratch directory, creates an output folder, runs the R script via `Rscript`, and compresses the results into a tarball for transfer back to the submit node. The tarball is built whether or not the R script succeeds (see the section on failed jobs below).

Usage

```
htc_gen_executable(
  output_file = NULL,
  r_script = NULL,
  data_files = NULL,
  results_folder = NULL,
  home_dir = "/home",
  mode = "single",
  set_executable = TRUE,
  verbose = FALSE,
  comments = FALSE,
  output = ".",
  config = NULL,
  path = "."
)
```

Arguments

`output_file`	A character string or `NULL`. Name of the shell script to write. Must end in `".sh"`. When `NULL` (the default), resolves to the `executable_file` recorded in the submission state by a previous `htc_gen_submit()` call (S-I3), so the name only

has to be typed once regardless of which of the two generators runs first. Falls back to "job.sh" if neither an explicit value nor a submission-state value is available. If the resolved value disagrees with an executable_file already in the submission state, warns rather than silently preferring one over the other.

r_script — A character string. Name of the R script that HTCondor will run, e.g. "analysis.R". Must be supplied explicitly – there is no default. If you used toolero::create_qmd(use_purl = TRUE), the script is the .R file produced by purl.R after rendering. The script itself is **not** baked into the container image – it travels to the execute node as an uploaded job input file (see transfer_input_files in htc_gen_submit()), so editing it only requires re-uploading and resubmitting, never a container rebuild. Because HTCondor's file transfer does not preserve subdirectories, only the file's basename is used inside the script; if you pass r_script = "R/analysis.R", make sure "analysis.R" (not the R/ path) is what actually lands in input_files for htc_gen_submit().

data_files — A character vector or NULL. Paths to data files baked into the container that should be passed to the R script as positional arguments. These are converted to absolute paths inside the container (e.g. "data-raw/sample.csv" becomes "/home/data-raw/sample.csv"). The R script receives them via commandArgs(trailingOnly = TRUE). Defaults to NULL.

results_folder — A character string or NULL. Name of the folder created in the scratch directory to hold job outputs before compression. When NULL (the default), resolves to config$project$conventions$output_dir when config is supplied (S-G5), falling back to "output", the output folder used across the toolero family, when neither is available. Note that only this folder is created: if your analysis writes to output/figures/, the R script must create that subfolder itself, which toolero::save_output() does and a bare ggsave() does not.

home_dir — A character string. The working directory inside the container where baked-in data_files live. Used to construct absolute paths for data file arguments only – it has no effect on where r_script is read from, since the R script is not baked into the image. Must match the home_dir used in containr::generate_dockerfile() for data_files. Defaults to "/home".

mode — A character string. Execution mode. "single" (the default) runs the R script with only the data file arguments (if any), producing a tarball named after the script alone. "multiple" also passes the subset filename as the first positional argument via ${1}, producing a per-job tarball that adds the subset's own stem, so analysis.R over adelie.csv gives analysis-adelie-results.tar.gz. Must match the mode used in htc_gen_submit(), which has to declare the same name in transfer_output_files.

set_executable — Logical. If TRUE, sets executable permissions on the generated script via Sys.chmod() so the file is ready to copy to the CHTC submit node without any additional steps. Defaults to TRUE. Set to FALSE if you prefer to manage permissions manually, in which case you must run chmod +x on the script before submitting your job.

verbose — Logical. If TRUE, prints progress messages as each section of the script is written. Defaults to FALSE.

`comments` Logical. If `TRUE`, annotates each section with an explanatory comment describing what the line does. Useful for researchers learning the HTCondor executable script conventions. Defaults to `FALSE`.

`output` A character string. Directory where the shell script will be written. Defaults to `"."` (current working directory).

`config` A named list as returned by `htc_config()`, or `NULL` (the default). When supplied with a `project` element (via `htc_config(project_config = )`), `config$project$conventions$` is used to default `results_folder` (S-G5). Not required – everything here can still be passed explicitly.

`path` A character string. Directory where the submission state (`htc-manifest.yml`) is read from and written to. Defaults to `"."` (the current working directory), matching the default used by `htc_upload()`, `htc_submit()`, and `htc_download()`. This is independent of `output`: if you write generated files to a subfolder with `output`, pass the same `path` explicitly to every function in the pipeline so they all find the same submission state.

Value

Called for its side effects. Writes a bash script to `file.path(output, output_file)` and sets executable permissions when `set_executable = TRUE`. Returns `invisible(NULL)`.

How file paths work inside the container

The generated script draws on two different sources for its inputs, and reads each one a different way:

The R script – `r_script` is not baked into the container image. It travels to the execute node as an uploaded job input file, listed in `transfer_input_files` by `htc_gen_submit()`. HTCondor's file transfer does not preserve subdirectories, so whatever you pass as `r_script` lands flat, by basename, in the scratch directory alongside the executable script itself. The `Rscript` line therefore refers to it by a bare relative name (e.g. `Rscript analysis.R`), not an absolute, `home_dir`-prefixed path. This is deliberate: editing the analysis script only requires re-uploading it and resubmitting the job, with no container rebuild or registry push in between.

Data files – `data_files` are the opposite case: they are baked into the container at build time by `containr::generate_dockerfile()` and live under `home_dir` (default `"/home"`). The `Rscript` line passes these as absolute paths (e.g. `/home/data-raw/sample.csv`) so they are found regardless of the working directory. Baking data in rather than uploading it keeps large or unchanging reference data out of every job's file transfer.

Writing – the script changes to HTCondor's scratch directory (`_CONDOR_SCRATCH_DIR`) before creating the output folder. This directory is writable and is where HTCondor looks for `transfer_output_files`. The R script writes outputs to `"output/"` using a relative path, which resolves to the scratch directory.

This separation means the R script stays portable – `"output/"` works in RStudio, in `quarto render`, and on HTCondor – while the `.sh` script handles the HTCondor-specific directory setup.

When the R script fails

The script runs under `set -euo pipefail`, so any failing command stops it – except the `Rscript` line, which is bracketed by `set +e` and `set -e` so that its exit status is recorded rather than fatal.

The results folder is then compressed as usual, and the script exits with the status R returned. A failed job therefore still sends back its tarball, holding whatever the analysis wrote before it stopped (and, with `toolero::save_output()`, the accumulator row recording the failed write), while HTCondor still sees a non-zero exit code and reports the job as failed rather than completed. Without this, a failed job would leave no tarball behind, and HTCondor would hold the job for a missing `transfer_output_files` entry instead of letting it finish.

Relationship to htc_gen_submit()

The executable script generated by `htc_gen_executable()` is the file referenced by the `executable` argument in `htc_gen_submit()`. The two functions should always use the same `mode`. In `"multiple"` mode, HTCondor passes each subset filename to the script as `${1}`, which is forwarded to the R script as a positional argument. The R script must be written to accept this argument – the recommended approach is `toolero::detect_execution_context()`:

```
context <- toolero::detect_execution_context()

input_file <- switch(context,
  interactive = "data-raw/sample.csv",
  quarto      = params$input_file,
  rscript     = commandArgs(trailingOnly = TRUE)[1]
)
```

Examples

```
# output writes the generated .sh file; path is where the submission state
# (htc-manifest.yml) gets read from and written to. The two are
# independent arguments (see @param path), so both must point at the
# same scratch directory here to keep the submission state out of the current
# working directory.
tmp <- tempdir()

# Single-job executable script with baked-in data
htc_gen_executable(
  r_script   = "R/analysis.R",
  data_files = "data-raw/sample.csv",
  output     = tmp,
  path       = tmp
)

# Multiple-job executable script
htc_gen_executable(
  r_script = "R/analysis.R",
  mode     = "multiple",
  output   = tmp,
  path     = tmp
)

# Custom names with annotations
htc_gen_executable(
  output_file = "run.sh",
```

```
  r_script    = "R/run-analysis.R",
  data_files  = c("data-raw/train.csv", "data-raw/test.csv"),
  comments    = TRUE,
  verbose     = TRUE,
  output      = tmp,
  path        = tmp
)
```

`htc_gen_submit` *Generate an HTCondor submit file for a containerized R job*

Description

`htc_gen_submit()` writes a ready-to-use HTCondor submit file (`.sub`) for running a containerized R job on an HTC cluster such as CHTC. It supports both single-job and multiple-job submission modes.

Usage

```
htc_gen_submit(
  output_file = "job.sub",
  container_image = NULL,
  executable = NULL,
  r_script = NULL,
  input_files = NULL,
  output_files = NULL,
  mode = "single",
  queue = 1L,
  queue_from = NULL,
  resources = "small",
  custom_resources = NULL,
  gpu = FALSE,
  gpu_options = NULL,
  verbose = FALSE,
  comments = FALSE,
  output = ".",
  config = NULL,
  path = "."
)
```

Arguments

`output_file` A character string. Name of the submit file to write. Must end in `".sub"`. Defaults to `"job.sub"`.

`container_image`

A character string. The container image to use, e.g. `"registry.doit.wisc.edu/netid/myimage"`. The `docker://` prefix is added automatically if not already present. Defaults to `NULL`, which writes a placeholder comment in the submit file.

executable — A character string or NULL. The shell script that HTCondor will run inside the container, e.g. "analysis.sh". When NULL (the default), resolves to the executable_file recorded in the submission state by a previous htc_gen_executable() call (S-I3), so the name only has to be typed once regardless of which of the two generators runs first. If neither an explicit value nor a submission-state value is available, writes a placeholder comment in the submit file instead. If the resolved value here disagrees with an executable_file already in the submission state, warns rather than silently preferring one over the other.

r_script — A character string. The R script the job runs, e.g. "R/analysis.R". Used only to derive the default output_files name, which must match the tarball htc_gen_executable() tells the job to build. This function never reads the executable script or the Dockerfile, so the script's name cannot be inferred and has to be given here. If omitted, the value recorded in the submission state by a previous htc_gen_executable() call is used; note that the documented workflow calls this function first, in which case there is nothing recorded yet. Ignored when output_files is supplied. Note that supplying r_script here does not transfer it – the script is not baked into the container image (see htc_gen_executable()), so its basename must also appear in input_files or HTCondor will not send it to the execute node; this function warns when it does not. Defaults to NULL.

input_files — A character vector. Files to transfer to the job's working directory before execution, given as paths on this machine, e.g. c("R/analysis.R", "data/lookup.csv"). This must include the R script named by r_script – the script travels to the execute node as an uploaded input file, not as part of the container image. The submission state keeps the paths as given, which is how htc_upload() finds the files; the submit file lists them by basename (transfer_input_files = analysis.R, lookup.csv), because htc_upload() sends every file flat into one directory on the submit node and HTCondor transfers them flat to the execute node. Two files that share a basename would overwrite each other there, so that is an error. In "multiple" mode, the per-job subset file is added automatically from the job manifest; use this argument for files shared across all jobs (e.g. the analysis script). Defaults to NULL.

output_files — A character vector. Files to transfer back from the job's working directory after execution. When not supplied, it is derived from r_script following the family convention <script stem>[-<subset stem>]-results.tar.gz, giving for example analysis-results.tar.gz in "single" mode and analysis-$Fn(file)-results.tar.gz in "multiple" mode. $Fn() is an HTCondor submit macro that strips a value's directory and extension, so a subset named adelie.csv yields analysis-adelie-results.tar.gz. Supplying this argument overrides the derivation entirely. Defaults to NULL.

mode — A character string. Submission mode. "single" (the default) submits one job. "multiple" submits one job per row in the job manifest supplied to queue_from, passing each subset file as a positional argument to the executable via arguments = $(file).

queue — A positive integer. Number of identical jobs to submit. Only used when mode = "single". Defaults to 1.

queue_from — A character string or NULL. Path to the job manifest produced by toolero::write_by_group(manifest = TRUE). Required when mode = "multiple", unless it can be resolved from config (S-G5): when NULL and config$project$conventions$split_dir is

set, defaults to `file.path(split_dir, "manifest.csv")` – `write_by_group()` always names its job manifest `manifest.csv`, so the convention's directory is enough to reconstruct the full path. The `file_path` column is extracted and written alongside the submit file as `subdatasets.csv`, which HTCondor reads to generate one job per subset file.

`resources` A character string. Compute resource preset. One of `"small"`, `"medium"`, `"large"`, or `"custom"` (requires `custom_resources`). Default preset values reflect CHTC recommendations and are loaded from `inst/extdata/htc-resources.yml`. A local `htc-resources.yml` in the working directory takes precedence over the package default, allowing per-project customization. Defaults to `"small"`.

`custom_resources` A named list. Required when `resources = "custom"`. Must contain `cpus` (integer), `memory` (character, e.g. `"8GB"`), and `disk` (character, e.g. `"4GB"`). Ignored when `resources` is not `"custom"`.

`gpu` Logical. If `TRUE`, adds GPU resource requests to the submit file. Defaults to `FALSE`.

`gpu_options` A named list or `NULL`. Fine-grained GPU options applied when `gpu = TRUE`. Supported keys: `request_gpus` (integer, default `1`), `want_gpu_lab` (logical, default `TRUE`), `min_capability` (numeric, e.g. `8.0` for A100; `NULL` to omit), `min_memory_mb` (integer in MB, e.g. `40000`; `NULL` to omit). When `gpu = TRUE` and `gpu_options = NULL`, CHTC defaults are used.

`verbose` Logical. If `TRUE`, prints progress messages as each section of the submit file is written. Defaults to `FALSE`.

`comments` Logical. If `TRUE`, annotates each section with an explanatory comment describing what the section does and how to use it. Defaults to `FALSE`.

`output` A character string. Directory where the submit file (and, in `"multiple"` mode, `subdatasets.csv`) will be written. Defaults to `"."` (current working directory).

`config` A named list as returned by `htc_config()`, or `NULL` (the default). When supplied with a `project` element (via `htc_config(project_config = )`), `config$project$conventions$` is used to default `queue_from` (S-G5). Not required – everything here can still be passed explicitly.

`path` A character string. Directory where the submission state (`htc-manifest.yml`) is read from and written to. Defaults to `"."` (the current working directory), matching the default used by `htc_upload()`, `htc_submit()`, and `htc_download()`. This is independent of `output`: if you write generated files to a subfolder with `output`, pass the same `path` explicitly to every function in the pipeline so they all find the same submission state.

Value

Called for its side effects. Writes an HTCondor submit file to `file.path(output, output_file)`. In `"multiple"` mode also writes `subdatasets.csv` to `output`. Returns `invisible(NULL)`.

Multiple-job mode and positional arguments

When `mode = "multiple"`, HTCondor passes each subset filename to the executable as a positional argument via `arguments = $(file)`. Your R script must be written to accept and use this argument.

The recommended approach is to use `toolero::detect_execution_context()` in your analysis script, which resolves the input file path correctly across interactive, Quarto, and Rscript execution contexts:

```
context <- toolero::detect_execution_context()

input_file <- switch(context,
  interactive = "data/penguins.csv",
  quarto      = params$input_file,
  rscript     = commandArgs(trailingOnly = TRUE)[1]
)

data <- readr::read_csv(input_file)
```

The typical workflow is:

1. Write and develop your analysis in `analysis.qmd` using `toolero::detect_execution_context()` for data loading.
2. Split your dataset with `toolero::write_by_group(manifest = TRUE)` to produce subset CSV files and a `manifest.csv`.
3. Strip `analysis.qmd` to `R/analysis.R` with `knitr::purl()`.
4. Call `htc_gen_submit(mode = "multiple", queue_from = "manifest.csv")` to produce the submit file and `subdatasets.csv`.
5. Copy `R/analysis.R`, the subset data files, `analysis.sub`, `analysis.sh`, and `subdatasets.csv` to CHTC and submit.

Resource presets

Resource presets are loaded at runtime from `inst/extdata/htc-resources.yml`. To customize presets for a specific project, copy that file to your project directory as `htc-resources.yml` and edit the values. `htc_gen_submit()` checks for a local `htc-resources.yml` in the working directory first, falling back to the package default if none is found.

submitr 0.1.0 named the file `htc-resources.yaml`. A local file by that name is still read, with a warning asking for it to be renamed, for this release only; the family writes YAML files with the `.yml` extension. When both names are present, `htc-resources.yml` is used and the old file is ignored, also with a warning.

Examples

```
# output writes the generated .sub file; path is where the submission state
# (htc-manifest.yml) gets read from and written to. The two are
# independent arguments (see @param path), so both must point at the
# same scratch directory here to keep the submission state out of the current
# working directory.
tmp <- tempdir()

# Single-job submit file with default resource preset
htc_gen_submit(output = tmp, path = tmp)
```

```
# Single-job submit file with medium resources and file transfer
htc_gen_submit(
  output_file     = "analysis.sub",
  container_image = "docker://registry.doit.wisc.edu/netid/myimage",
  executable      = "analysis.sh",
  r_script        = "R/analysis.R",
  input_files     = "R/analysis.R",
  resources       = "medium",
  output          = tmp,
  path            = tmp
)

# Annotated submit file useful for learning HTCondor syntax
htc_gen_submit(
  output_file = "annotated.sub",
  comments    = TRUE,
  verbose     = TRUE,
  output      = tmp,
  path        = tmp
)

# Custom resource request
htc_gen_submit(
  resources        = "custom",
  custom_resources = list(cpus = 2, memory = "8GB", disk = "4GB"),
  output           = tmp,
  path             = tmp
)

## Not run:
# Multiple-job submit file driven by a write_by_group() job manifest
htc_gen_submit(
  output_file     = "analysis.sub",
  container_image = "docker://registry.doit.wisc.edu/netid/myimage",
  executable      = "analysis.sh",
  input_files     = "analysis.R",
  mode            = "multiple",
  queue_from      = "data/manifest.csv",
  resources       = "medium",
  output          = "."
)

## End(Not run)
```

`htc_release` *Release held HTCondor jobs*

Description

`htc_release()` connects to an HTC submit node via SSH and runs `condor_release` to un-pause

jobs that HTCondor has held (S-G2) – typically after fixing whatever caused the hold, such as raising a resource request reported by `htc_status()`'s hold-reason output.

Usage

```
htc_release(
  cluster_id = NULL,
  config = NULL,
  dry_run = FALSE,
  verbose = FALSE,
  path = "."
)
```

Arguments

`cluster_id`	An integer, character string, or `NULL`. The cluster ID to release, e.g. `6302860`. When `NULL` (the default), resolves to the cluster ID recorded in the submission state by the most recent `htc_submit()` call. Unlike `htc_status()`, `htc_release()` never falls back to "all of my jobs": a bare `condor_release` with no arguments releases every held job you have on the submit node, not just the ones from this project, so the function errors instead of guessing when no cluster ID is available anywhere.
`config`	A named list as returned by `htc_config()`. Must contain `username` and `server`. If `NULL` (the default), uses the session config set by `htc_start()`. If no session config is set, the function errors with instructions.
`dry_run`	Logical. If `TRUE`, prints the SSH command that would be executed without running it. Defaults to `FALSE`.
`verbose`	Logical. If `TRUE`, prints progress messages. Defaults to `FALSE`.
`path`	A character string. Directory holding the submission state (`htc-manifest.yml`), consulted only when `cluster_id` is `NULL`. Defaults to `"."`, matching the default used elsewhere in the family.

Value

The `condor_release` output as a character vector, returned invisibly.

Workflow

```
cfg <- htc_config()
htc_status(cluster_id = 6302860, config = cfg)
# ... shows job 6302860.0 held; hold reason printed below the table ...
# ... fix the resource request that caused the hold, then:
htc_release(cluster_id = 6302860, config = cfg)
```

See Also

`htc_cancel()` to remove a job instead of releasing it, and `htc_status()` to see hold reasons.

Examples

```
# Preview the SSH command without connecting to CHTC
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
htc_release(cluster_id = 6302860, config = cfg, dry_run = TRUE)


## Not run:
cfg <- htc_config()
htc_release(cluster_id = 6302860, config = cfg)

## End(Not run)
```

---

htc_ssh_setup *Configure SSH connection reuse (ControlMaster) for CHTC*

Description

`htc_ssh_setup()` writes the ControlMaster block described in `htc_config()`'s own setup guidance to `~/.ssh/config` and creates the connections directory it references, so that reuse can be set up without leaving R (S-G3). Every function in this package that opens an SSH connection – `htc_upload()`, `htc_submit()`, `htc_status()`, `htc_download()`, `htc_cancel()`, and `htc_release()` – benefits, since each currently opens a fresh connection per call and triggers a separate Duo MFA prompt without this.

Usage

```
htc_ssh_setup(
  ssh_config_path = "~/.ssh/config",
  connections_dir = "~/.ssh/connections",
  host_pattern = "*.chtc.wisc.edu",
  dry_run = FALSE
)
```

Arguments

`ssh_config_path`	A character string. Path to the SSH client config file to append to. Defaults to `"~/.ssh/config"`.
`connections_dir`	A character string. Path to the directory `ControlPath` will use for its control sockets. Defaults to `"~/.ssh/connections"`. Written into the config file exactly as given (so a leading `~` is expanded by `ssh` itself at connection time, not by R), and created on disk if it does not already exist.
`host_pattern`	A character string. The `Host` pattern the block applies to. Defaults to `"*.chtc.wisc.edu"`, matching every CHTC submit node. Narrow this if you only want reuse for one specific host.
`dry_run`	Logical. If `TRUE`, prints what would be added and created without writing anything. Defaults to `FALSE`.

Value

Called for its side effects. Returns `invisible(NULL)`.

What this does not do

If a `Host` block matching `host_pattern` already exists in `ssh_config_path`, `htc_ssh_setup()` leaves the file untouched and says so – it never edits or replaces an existing block, since the existing one may have been customized deliberately. Remove or edit it by hand first if you want `htc_ssh_setup()` to write a fresh one.

Windows

`ControlPath` relies on Unix domain sockets, which native Windows OpenSSH (as shipped with Windows 10/11) has historically supported inconsistently across versions. `htc_ssh_setup()` still writes the block on Windows – it may simply work, particularly on a current OpenSSH release – but if `ssh` subsequently errors mentioning `ControlPath` or a socket, connection reuse is not supported on this system: remove the added block and expect a Duo MFA prompt on every call instead. OpenSSH bundled with WSL or with Git Bash is more likely to support this than the native Windows client.

See Also

`htc_config()`, whose own first-run message describes this same block for manual setup.

Examples

```
# Preview without touching any files
htc_ssh_setup(dry_run = TRUE)


## Not run:
htc_ssh_setup()

# Apply only to one specific submit node
htc_ssh_setup(host_pattern = "ap2002.chtc.wisc.edu")

## End(Not run)
```

`htc_start` *Start an HTC session*

Description

`htc_start()` calls `htc_config()` to read or create the connection configuration, then stores the result as a session-level option so that subsequent `htc_*()` functions can use it without requiring an explicit `config` argument on every call.

Usage

```
htc_start(...)
```

Arguments

`...`	Arguments passed to `htc_config()`. Common arguments include `username`, `server`, `path`, `overwrite`, `check_server` (turns off the SSH reachability probe for scripted or unattended use), and `project_config` (folds a `_toolero.yml` project's folders and conventions into the returned config under `config$project`).

Details

After calling `htc_start()`, functions like `htc_upload()`, `htc_submit()`, `htc_status()`, and `htc_download()` will automatically use the stored configuration when `config = NULL` (the default). You can still pass `config` explicitly to any function to override the session config.

The session config is stored via `options(submitr.config = ...)` and is cleared automatically when the R session ends. To clear it manually, call `options(submitr.config = NULL)`.

The submission state built up by `htc_gen_submit()`, `htc_gen_executable()`, and `htc_submit()` is persisted to disk separately (see `htc_upload()`/`htc_download()`), so calling `htc_start()` again – for example, in a new R session – does not discard it.

Value

Invisibly returns the config list (same as `htc_config()`).

Examples

```
## Not run:
# Start a session -- all subsequent htc_*() calls use this config
htc_start()

# Now these work without config = cfg
htc_upload(files = c("job.sub", "job.sh"))
htc_submit(submit_file = "job.sub")
htc_status(cluster_id = 6351616)
htc_download(files = "*.tar.gz")

# You can still override for a specific call
other_cfg <- htc_config(path = "other-project/")
htc_upload(files = "job.sub", config = other_cfg)

# Clear the session config manually
options(submitr.config = NULL)

## End(Not run)
```

`htc_status` *Check the status of submitted HTCondor jobs*

Description

`htc_status()` connects to an HTC submit node via SSH and runs `condor_q` to report the status of jobs in the queue. By default it shows all of your jobs. Optionally filter by cluster ID to monitor a specific submission.

Usage

```
htc_status(
  cluster_id = NULL,
  config = NULL,
  watch = FALSE,
  interval = 60L,
  dry_run = FALSE,
  verbose = FALSE,
  path = ".",
  show_hold_reason = TRUE
)
```

Arguments

`cluster_id`	An integer, character string, or `NULL`. The cluster ID returned by `htc_submit()`, e.g. `6302860`. When `NULL` (the default), resolves to the cluster ID recorded in the submission state by the most recent `htc_submit()` call; if the submission state holds no value either, shows all of your jobs currently in the queue instead. Required (directly or via the submission state) when `watch = TRUE`.
`config`	A named list as returned by `htc_config()`. Must contain `username` and `server`. If `NULL` (the default), uses the session config set by `htc_start()`. If no session config is set, the function errors with instructions.
`watch`	Logical. If `TRUE`, polls the queue repeatedly at `interval` seconds until all jobs in `cluster_id` have completed. Requires `cluster_id` to be supplied. Defaults to `FALSE`.
`interval`	A positive integer. Number of seconds to wait between polls when `watch = TRUE`. Defaults to `60`.
`dry_run`	Logical. If `TRUE`, prints the SSH command that would be executed without running it. Defaults to `FALSE`.
`verbose`	Logical. If `TRUE`, prints progress messages. Defaults to `FALSE`.
`path`	A character string. Directory holding the submission state (`htc-manifest.yml`), consulted only when `cluster_id` is `NULL`. Defaults to `"."`, matching the default used by `htc_upload()`, `htc_submit()`, and `htc_download()`. If you passed a non-default `path` to `htc_submit()`, pass that same directory here.

show_hold_reason

Logical. If TRUE (the default), every poll automatically follows up with condor_q -hold and prints the result whenever it shows an actual held job – nothing is printed when nothing is held (S-G2). This is the single most common cause of newcomer confusion – a job held for exceeding its memory or disk request otherwise gives no clue why without a separate manual query. Set to FALSE to skip the follow-up query entirely (for example, in a tight watch = TRUE polling loop where the extra round trip is unwelcome).

Details

When watch = TRUE, htc_status() polls the queue repeatedly at a fixed interval until all jobs in the cluster have completed, printing a timestamped snapshot after each poll.

Value

Called for its side effects. Prints the condor_q output (and, when applicable, hold reasons) to the console. Returns the most recent condor_q output invisibly as a character vector.

Job status codes

HTCondor reports each job's status with a single letter:

Code	Meaning
I	Idle – waiting for a matching execute node
R	Running – currently executing
H	Held – paused, usually due to an error
C	Completed – finished successfully
X	Removed – cancelled
S	Suspended

Jobs disappear from condor_q once they complete and their output has been transferred back to the submit node. Use htc_download() to retrieve completed job output.

A held (H) job is not a lost cause: htc_status() surfaces the hold reason automatically (see show_hold_reason), and once the underlying problem is fixed – most often the resource request – htc_release() resumes it without resubmitting. To abandon a job entirely instead, use htc_cancel().

Workflow

```
cfg <- htc_config()

# One-shot status check
htc_status(config = cfg)

# Monitor a specific cluster until completion
htc_status(cluster_id = 6302860, config = cfg, watch = TRUE)
```

SSH connection reuse

Each poll in watch mode opens a new SSH connection. Configuring ControlMaster in your `~/.ssh/config` (see `htc_config()`) is strongly recommended when using `watch = TRUE` to avoid repeated Duo MFA prompts.

See Also

`htc_cancel()` to remove a job, and `htc_release()` to resume a held one after fixing what caused the hold.

Examples

```
# Preview the SSH command without connecting to CHTC
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
htc_status(config = cfg, dry_run = TRUE)

# Preview with a specific cluster ID
htc_status(cluster_id = 6302860, config = cfg, dry_run = TRUE)


## Not run:
# All remaining examples require a live CHTC connection
cfg <- htc_config()

# Check all your jobs
htc_status(config = cfg)

# Check a specific cluster
htc_status(cluster_id = 6302860, config = cfg)

# Watch a cluster until all jobs complete (polls every 60 seconds)
htc_status(cluster_id = 6302860, config = cfg, watch = TRUE)

# Watch with a shorter polling interval
htc_status(cluster_id = 6302860, config = cfg, watch = TRUE, interval = 30)

## End(Not run)
```

`htc_submit` *Submit an HTCondor job from a remote submit node*

Description

`htc_submit()` connects to an HTC submit node via SSH and runs `condor_submit` on a submit file that has already been uploaded with `htc_upload()`. It changes into the remote directory before submitting so that relative paths in the submit file resolve correctly.

Usage

```
htc_submit(
  submit_file = NULL,
  remote_path = NULL,
  config = NULL,
  dry_run = FALSE,
  verbose = FALSE,
  path = "."
)
```

Arguments

submit_file	A character string or NULL. Name of the submit file on the remote node, e.g. "job.sub". Must end in ".sub". When NULL (the default), resolves to the submit_file recorded in the submission state by htc_gen_submit(), falling back to "job.sub" if the submission state holds no value.
remote_path	A character string or NULL. The directory on the submit node where the submit file was uploaded. When NULL (the default), resolves to the remote_path recorded in the submission state by the preceding call to htc_upload(), falling back to "~/" if the submission state holds no value.
config	A named list as returned by htc_config(). Must contain username and server. If NULL (the default), uses the session config set by htc_start(). If no session config is set, the function errors with instructions.
dry_run	Logical. If TRUE, prints the SSH command that would be executed without running it. Useful for verifying the command before submitting. Defaults to FALSE.
verbose	Logical. If TRUE, prints progress messages and the condor_submit output. Defaults to FALSE.
path	A character string. Directory holding the submission state (htc-manifest.yml), where the cluster ID and remote_path are recorded on a successful submission. Defaults to ".". If you passed a non-default output or path to htc_gen_submit(), pass that same directory here, or the cluster ID will be written to a second, separate htc-manifest.yml and htc_download() will not find it alongside the job metadata it needs.

Value

The cluster ID assigned by HTCondor as a character string, returned invisibly. Pass it directly to htc_status() to monitor job progress. Returns invisible(NULL) if the cluster ID cannot be parsed from the condor_submit output.

Workflow

htc_submit() is the second system-facing step in the submitr workflow. Call it after uploading your files with htc_upload(). The returned cluster ID can be passed directly to htc_status().

```
cfg <- htc_config()
```

```
htc_upload(
  files  = c("job.sub", "job.sh", "analysis.R"),
  config = cfg
)

cluster_id <- htc_submit(submit_file = "job.sub", config = cfg)
htc_status(cluster_id = cluster_id, config = cfg, watch = TRUE)
```

Why `remote_path` must match `htc_upload()`

`htc_submit()` runs `cd remote_path && condor_submit submit_file` on the submit node. HTCondor resolves all paths in the submit file relative to the directory where `condor_submit` is called. If `remote_path` does not match the directory where files were uploaded, HTCondor will not find the executable, input files, or output destinations and the job will fail.

SSH connection reuse

Each call to `htc_submit()` opens a new SSH connection. If you have not configured ControlMaster in your `~/.ssh/config`, this will trigger a Duo MFA prompt. Run `htc_config()` for setup guidance.

Examples

```
# Preview the SSH command without connecting to CHTC
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
htc_submit(submit_file = "job.sub", config = cfg, dry_run = TRUE)


## Not run:
# All remaining examples require a live CHTC connection
cfg <- htc_config()

# Submit using default remote path
htc_submit(submit_file = "job.sub", config = cfg)

# Submit from a specific remote directory
htc_submit(
  submit_file = "analysis.sub",
  remote_path = "~/projects/penguins/",
  config      = cfg
)

# Submit with verbose output to see condor_submit response
htc_submit(
  submit_file = "job.sub",
  config      = cfg,
  verbose     = TRUE
)

## End(Not run)
```

htc_upload *Upload files to an HTC submit node*

Description

`htc_upload()` copies one or more local files or directories to a directory on an HTC submit node via `scp`. It is the first step in the job submission workflow – files must be present on the submit node before `htc_submit()` can run `condor_submit`.

Usage

```
htc_upload(
  files = NULL,
  remote_path = NULL,
  config = NULL,
  dry_run = FALSE,
  verbose = FALSE,
  path = ".",
  check = FALSE
)
```

Arguments

`files`	A character vector or `NULL`. One or more local file paths or directory paths to copy to the submit node. A single file, a vector of files, and a directory path are all accepted. Directories are copied recursively. When `NULL` (the default), the function resolves the files to upload from the submission state built up by `htc_gen_submit()` and `htc_gen_executable()`: the submit file, the executable script, any shared input files, and – in `"multiple"` mode – `subdatasets.csv` and the individual subset data files.
`remote_path`	A character string or `NULL`. The destination directory on the submit node. When `NULL` (the default), resolves to the `remote_path` recorded in the submission state by a previous call to `htc_upload()`, falling back to `"~/"` if the submission state holds no value. On a successful (non-`dry_run`) upload, the resolved value is recorded back to the submission state, so `htc_submit()` and `htc_download()` can pick it up automatically without retyping it.
`config`	A named list as returned by `htc_config()`. Must contain `username` and `server`. If `NULL` (the default), uses the session config set by `htc_start()`. If no session config is set, the function errors with instructions.
`dry_run`	Logical. If `TRUE`, prints the `scp` command that would be executed without running it. Useful for verifying the command before transferring files. Defaults to `FALSE`.
`verbose`	Logical. If `TRUE`, prints progress messages. Defaults to `FALSE`.
`path`	A character string. Directory holding the submission state (`htc-manifest.yml`), consulted only when `files` is `NULL`. Defaults to `"."`, which matches the generator functions' own default. If you passed a non-default `output` or `path`

to htc_gen_submit() and htc_gen_executable(), pass that same directory here.

check — Logical. If TRUE, runs htc_check() before uploading and aborts if it finds an "error"-level issue (a missing file, or a "multiple"-mode subset mismatch) – catching it here rather than an hour later as a held job on the cluster (S-G4). Warning-level issues (a resource request that looks large, an unconfirmed image) are reported but do not block the upload. Defaults to FALSE.

Value

Called for its side effects. Returns invisible(NULL).

Workflow

htc_upload() is the first system-facing step in the submitr workflow. Call it after generating your submit file and executable script with htc_gen_submit() and htc_gen_executable(), and before calling htc_submit().

The typical sequence relies on automatic resolution from the submission state, so files can usually be omitted:

```
cfg <- htc_config()

htc_gen_submit(executable = "job.sh", r_script = "R/analysis.R",
               input_files = "R/analysis.R")
htc_gen_executable(r_script = "R/analysis.R")

htc_upload(config = cfg)

htc_submit(submit_file = "job.sub", config = cfg)
```

Pass files explicitly to upload a specific set of files instead:

```
htc_upload(
  files  = c("job.sub", "job.sh", "analysis.R", "data.csv"),
  config = cfg
)
```

SSH connection reuse

Each call to htc_upload() opens a new SSH connection. If you have not configured ControlMaster in your ~/.ssh/config, this will trigger a Duo MFA prompt. Run htc_config() for setup guidance.

Examples

```
# Preview the scp command without connecting to CHTC
cfg <- list(username = "netid", server = "ap2002.chtc.wisc.edu")
tmp <- tempfile(fileext = ".sub")
writeLines("queue 1", tmp)
```

```
htc_upload(files = tmp, config = cfg, dry_run = TRUE)


## Not run:
# All remaining examples require a live CHTC connection
cfg <- htc_config()

# Resolve files automatically from the submission state
htc_gen_submit(executable = "job.sh", r_script = "R/analysis.R",
               input_files = "R/analysis.R")
htc_gen_executable(r_script = "R/analysis.R")
htc_upload(config = cfg)

# Upload a single file
htc_upload(files = "job.sub", config = cfg)

# Upload multiple files
htc_upload(
  files  = c("job.sub", "job.sh", "analysis.R"),
  config = cfg
)

# Upload a directory
htc_upload(files = "jobs/", config = cfg)

# Upload to a specific remote directory
htc_upload(
  files       = c("job.sub", "job.sh"),
  remote_path = "~/projects/penguins/",
  config      = cfg
)

## End(Not run)
```

Index

htc_cancel, 2
htc_cancel(), *23, 24*, *28, 29*
htc_check, 4
htc_check(), *33*
htc_collect, 6
htc_config, 8
htc_config(), *3*, *12, 13*, *16*, *20*, *23–27*, *29–33*
htc_download, 11
htc_download(), *6*, *16*, *20*, *24*, *26–28*, *30*, *32*
htc_gen_executable, 14
htc_gen_executable(), *4, 5*, *7*, *10*, *12*, *19*, *26*, *32, 33*
htc_gen_submit, 18
htc_gen_submit(), *4, 5*, *10*, *12*, *14–17*, *26*, *30*, *32, 33*
htc_release, 22
htc_release(), *3*, *24*, *28, 29*
htc_ssh_setup, 24
htc_start, 25
htc_start(), *3*, *12*, *23*, *27*, *30*, *32*
htc_status, 27
htc_status(), *3*, *11*, *13*, *23, 24*, *26*, *28*, *30*
htc_submit, 29
htc_submit(), *3*, *12*, *16*, *20*, *23, 24*, *26, 27*, *32, 33*
htc_upload, 32
htc_upload(), *5*, *12*, *16*, *19, 20*, *24*, *26, 27*, *29, 30*