

# Package ‘bsub’

April 8, 2026

**Type** Package

**Title** Submitter and Monitor of the 'LSF Cluster'

**Version** 2.0.7

**Date** 2026-04-08

**Depends** R (>= 4.0.0)

**Imports** GlobalOptions (>= 0.1.1), GetoptLong (>= 0.1.8), digest, utils, stats, clisymbols, crayon, grDevices, graphics, codetools, ssh, igraph

**Suggests** knitr, DT (>= 0.13), shiny (>= 1.0.0), shinyjs, graph, ggplot2, rmarkdown, markdown, DiagrammeR

**biocViews** Software, Infrastructure

**Description** It submits R code/R scripts/shell commands to 'LSF cluster' (<[https://en.wikipedia.org/wiki/Platform\\_LSF](https://en.wikipedia.org/wiki/Platform_LSF)>, the 'bsub' system) without leaving R. There is also an interactive 'shiny' application for monitoring job status.

**VignetteBuilder** knitr

**URL** <https://github.com/jokergoo/bsub>

**License** MIT + file LICENSE

**NeedsCompilation** no

**SystemRequirements** Platform LSF, bsub

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Author** Zuguang Gu [aut, cre] (ORCID: <<https://orcid.org/0000-0002-7395-8709>>)

**Maintainer** Zuguang Gu <guzuguang@suat-sz.edu.cn>

**Repository** CRAN

**Date/Publication** 2026-04-08 09:40:14 UTC

## Contents

bconf	2
bjobs	3
bjobs_barplot	5
bjobs_reset_timestamp	6
bkill	6
bsub_chunk	7
bsub_opt	10
config_bsub	12
formatFileSize	12
job_dependency_all	13
job_log	14
job_rerun	15
job_status_by_name	15
list_dump_files	16
list_temp_files	17
monitor	17
print.bjobs	18
random_job	18
retrieve_var	19
run_cmd	20
ssh_connect	20

<b>Index</b>	<b>22</b>
--------------	-----------

---

bconf	<i>Print current configurations</i>
-------	-------------------------------------

---

### Description

Print current configurations

### Usage

bconf

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

### Arguments

x	A bconf object.
...	Other parameters.

### Format

An object of class bconf of length 1.

**Details**

This function is only for printing. Use `bsub_opt()` to change configurations.

You simply type `bconf` (without the brackets) in the interactive R console.

**Value**

A `bconf` object.

**Examples**

```
bconf
```

---

bjobs	<i>Summary of jobs</i>
-------	------------------------

---

**Description**

Summary of jobs

**Usage**

```
bjobs(
  status = c("RUN", "PEND"),
  max = Inf,
  filter = NULL,
  print = TRUE,
  job_id = NULL
)

bjobs_raw(fields = "jobid stat job_name queue")

brecent(max = 20, filter = NULL)

bjobs_running(max = Inf, filter = NULL)

bjobs_pending(max = Inf, filter = NULL)

bjobs_done(max = Inf, filter = NULL)

bjobs_exit(max = Inf, filter = NULL)
```

**Arguments**

<code>status</code>	Status of the jobs. Use "all" for all status
<code>max</code>	Maximal number of recent jobs.
<code>filter</code>	Regular expression on job names.

print	Whether to print the table.
job_id	A single job ID, internally used.
fields	Supported output fields, check <a href="https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=information-customize-job-output">https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=information-customize-job-output</a> . The value can be a vector of field names or a single string that already includes output fields separated by space.

## Details

There is an additional column "RECENT" which is the order for the job with the same name. 1 means the most recent job.

You can directly type `bjobs` without parentheses which runs `bjobs` with defaults.

- `brecent` shows the most recent.
- `bjobs_done` shows the "DONE" jobs.
- `bjobs_exit` shows the "EXIT" jobs.
- `bjobs_pending` shows the "PEND" jobs.
- `bjobs_running` shows the "RUN" jobs.

`bjobs_raw()` returns the table from the original `bsubs -a -o '...'` call.

## Value

A data frame with selected job summaries.

## Examples

```
## Not run:
bjobs # this is the same as bjobs()
bjobs() # all running and pending jobs
bjobs(status = "all") # all jobs
bjobs(status = "RUN") # all running jobs, you can also use `bjobs_running`
bjobs(status = "PEND") # all pending jobs, you can also use `bjobs_pending`
bjobs(status = "DONE") # all done jobs, you can also use `bjobs_done`
bjobs(status = "EXIT") # all exit jobs, you can also use `bjobs_exit`
bjobs(status = "all", max = 20) # last 20 jobs
bjobs(status = "DONE", filter = "example") # done jobs with name `.*example.*`

## End(Not run)
## Not run:
brecent # this is the same as `brecent()`
brecent() # last 20 jobs (from all status)
brecent(max = 50) # last 50 jobs
brecent(filter = "example") # last 20 jobs with name `.*example.*`

## End(Not run)
## Not run:
bjobs_running # this is the same as `bjobs_running()`
bjobs_running() # all running jobs
bjobs_running(max = 50) # last 50 running jobs
```

```
bjobs_running(filter = "example") # running jobs with name ".*example.*"

## End(Not run)
## Not run:
bjobs_pending # this is the same as `bjobs_pending()`
bjobs_pending() # all pending jobs
bjobs_pending(max = 50) # last 50 pending jobs
bjobs_pending(filter = "example") # pending jobs with name ".*example.*"

## End(Not run)
## Not run:
bjobs_done # this is the same as `bjobs_done()`
bjobs_done() # all done jobs
bjobs_done(max = 50) # last 50 done jobs
bjobs_done(filter = "example") # done jobs with name ".*example.*"

## End(Not run)
## Not run:
bjobs_exit # this is the same as `bjobs_exit()`
bjobs_exit() # all exit jobs
bjobs_exit(max = 50) # last 50 exit jobs
bjobs_exit(filter = "example") # exit jobs with name ".*example.*"

## End(Not run)
```

---

bjobs\_barplot

*Visualize statistics of jobs*

---

## Description

Visualize statistics of jobs

## Usage

```
bjobs_barplot(
  status = c("RUN", "EXIT", "PEND", "DONE"),
  filter = NULL,
  job_tb = NULL
)

bjobs_timeline(
  status = c("RUN", "EXIT", "PEND", "DONE"),
  filter = NULL,
  job_tb = NULL
)
```

## Arguments

**status**                    Status of the jobs. Use "all" for all status.

filter	Regular expression to filter on job names.
job_tb	A data frame returned by <code>bjobs()</code> . Only internally used.

### Details

`bjobs_barplot()` draws barplots of number of jobs per day.

`bjobs_timeline()` draws segments of duration of jobs. In the plot, each segment represents a job and the width of the segment correspond to its duration.

---

`bjobs_reset_timestamp` *Clear job history*

---

### Description

Clear job history

### Usage

```
bjobs_reset_timestamp()
```

### Details

It sets a timestamp to only show jobs after it.

---

`bkill` *Kill jobs*

---

### Description

Kill jobs

### Usage

```
bkill(job_id, filter = NULL)
```

### Arguments

job_id	A vector of job IDs or a data frame returned by <code>bjobs()</code> .
filter	Regular expression on job names (only the running and pending jobs). It is only used when <code>job_id</code> is not set.

**Examples**

```
## Not run:
job_id = c(10000000, 10000001, 10000002) # job ids can be get from `bjobs()`
bkill(job_id)
# kill all jobs (running and pending) of which the names contain "example"
bkill(filter = "example")

## End(Not run)
```

---

bsub\_chunk

*Submit jobs*

---

**Description**

Submit jobs

**Usage**

```
bsub_chunk(
  code,
  name = NULL,
  packages = bsub_opt$packages,
  image = bsub_opt$image,
  variables = character(),
  share = character(),
  working_dir = bsub_opt$working_dir,
  hours = 1,
  memory = 1,
  cores = 1,
  R_version = bsub_opt$R_version,
  temp_dir = bsub_opt$temp_dir,
  output_dir = bsub_opt$output_dir,
  dependency = NULL,
  enforce = bsub_opt$enforce,
  local = bsub_opt$local,
  script = NULL,
  start = NULL,
  end = NULL,
  save_var = FALSE,
  sh_head = bsub_opt$sh_head,
  ask = TRUE
)

bsub_script(
  script,
  argv = "",
  name = NULL,
```

```

working_dir = bsub_opt$working_dir,
hours = 1,
memory = 1,
cores = 1,
R_version = bsub_opt$R_version,
temp_dir = bsub_opt$temp_dir,
output_dir = bsub_opt$output_dir,
dependency = NULL,
enforce = bsub_opt$enforce,
local = bsub_opt$local,
sh_head = bsub_opt$sh_head,
ask = TRUE,
...
)

bsub_cmd(
  cmd,
  sh = NULL,
  name = NULL,
  hours = 1,
  memory = 1,
  cores = 1,
  temp_dir = bsub_opt$temp_dir,
  output_dir = bsub_opt$output_dir,
  dependency = NULL,
  enforce = bsub_opt$enforce,
  local = bsub_opt$local,
  env_var = NULL,
  sh_head = bsub_opt$sh_head,
  ask = TRUE,
  ...
)

```

### Arguments

code	The code chunk, it should be embraced by {}.
name	If name is not specified, an internal name calculated by <code>digest::digest()</code> on the chunk is automatically assigned.
packages	A character vector with package names that will be loaded before running the script. There is a special name <code>_in_session_</code> that loads all the packages loaded in current R session.
image	A character vector of <code>.RData/.rda</code> files that will be loaded before running the script. When image is set to TRUE, all variables in <code>.GlobalEnv</code> will be saved into a temporary file and all attached packages will be recorded. The temporary files will be removed after the job is finished.
variables	A character vector of variable names that will be loaded before running the script. There is a special name <code>_all_functions_</code> that saves all functions defined in the global environment.

share	A character vector of variables names for which the variables are shared between jobs. Note the temporary .RData files are not deleted automatically.
working_dir	The working directory.
hours	Running time of the job.
memory	Memory usage of the job. It is measured in GB.
cores	Number of cores.
R_version	R version.
temp_dir	Path of temporary folder where the temporary R/bash scripts will be put.
output_dir	Path of output folder where the output/flag files will be put.
dependency	A vector of job IDs that current job depends on.
enforce	If a flag file for the job is found, whether to enforce to rerun the job.
local	Run job locally (not submitting to the LSF cluster)?
script	In <code>bsub_chunk()</code> , it is the path of a script where code chunks will be extracted and sent to the cluster. It is always used with <code>start</code> and <code>end</code> arguments. In <code>bsub_script()</code> , it is the path of the R script to submit.
start	A numeric vector that contains line indices of the starting code chunk or a character vector that contain regular expression to match the start of code chunks.
end	Same setting as <code>start</code> .
save_var	Whether save the last variable in the code chunk? Later the variable can be retrieved by <code>retrieve_var()</code> .
sh_head	Commands that are written as head of the sh script.
ask	Whether to promote.
argv	A string of command-line arguments.
...	Command-line arguments can also be specified as name-value pairs.
cmd	A single-line command.
sh	Path of the bash script.
env_var	Environment variables. It should be a named vector. Note environment variables can also be directly set in <code>sh_head</code> .

### Details

`job_chunk()` submits R code chunk.

`job_script()` submits R script with command-line arguments.

`job_cmd()` submits general bash commands.

### Value

A job ID.

**Examples**

```
## Not run:
bsub_chunk(name = "example", memory = 10, hours = 10, cores = 4,
{
  Sys.sleep(5)
})

# the R version is defined in bsub_opt$R_version
bsub_script("/path/of/foo.R", name = ..., memory = ..., cores = ..., ...)
# with command-line arguments
bsub_script("/path/of/foo.R", argv = "--a 1 --b 3", ...)

# put all arguments also in the command
bsub_cmd("some-tool -arg1 1 -arg2 2", name = ..., memory = ..., cores = ..., ...)

## End(Not run)
```

---

bsub\_opt

---

*Parameters for bsub*


---

**Description**

Parameters for bsub

**Usage**

```
bsub_opt(..., RESET = FALSE, READ.ONLY = NULL, LOCAL = FALSE, ADD = FALSE)
```

**Arguments**

...	Arguments for the parameters, see "details" section
RESET	reset to default values
READ.ONLY	please ignore
LOCAL	please ignore
ADD	please ignore

**Details**

There are following global parameters:

- `packages`: A character vector with package names that will be loaded before running the script.
- `image`: A character vector of RData/rda files that will be loaded before running the script.
- `temp_dir`: Path of temporary folder where the temporary R/bash scripts will be put.
- `output_dir`: Path of output folder where the output/flag files will be put.
- `enforce`: If a flag file for the job is found, whether to enforce to rerun the job.

- `R_version`: The version of R.
- `working_dir`: The working directory.
- `ignore`: Whether ignore `bsub_chunk`, `bsub_script` and `bsub_cmd`.
- `local`: Run job locally (not submitting to the LSF cluster)?
- `call_Rscript`: How to call `Rscript` by specifying an R version number.
- `submission_node`: A list of node names for submitting jobs.
- `login_node`: This value basically is the same as `submission_node` unless the login nodes are different from submission nodes.
- `sh_head`: Commands that are written as head of the sh script.
- `user`: Username on the submission node.
- `group`: The user group
- `ssh_envir`: The commands for setting bash environment for successfully running `bjobs`, `bsub`, ...
- `bsub_template`: Template for constructing `bsub` command.
- `parse_time`: A function that parses time string from the LSF `bjobs` command to a `POSIXct` object.
- `verbose`: Whether to print more messages.

`ssh_envir` should be properly set so that LSF binaries such as `bsub` or `bjobs` can be properly found. There are some environment variables initialized when logging in the bash terminal while they are not initialized with the ssh connection. Thus, some environment variables should be manually set.

An example for `ssh_envir` is as follows. The `LSF_ENVDIR` and `LSF_SERVERDIR` should be defined and exported.

```
c("source /etc/profile",
  "export LSF_ENVDIR=/opt/lsf/conf",
  "export LSF_SERVERDIR=/opt/lsf/10.1/linux3.10-glibc2.17-x86_64/etc")
```

The values of these two variables can be obtained by entering following commands in your bash terminal (on the submission node):

```
echo $LSF_ENVDIR
echo $LSF_SERVERDIR
```

The time strings by LSF `bjobs` command might be different for different configurations. The **bsub** package needs to convert the time strings to `POSIXct` objects for calculating the time difference. Thus, if the default time string parsing fails, users need to provide a user-defined function and set with `parse_time` option in `bsub_opt`. The function accepts a vector of time strings and returns a `POSIXct` object. For example, if the time string returned from `bjobs` command is in a form of `Dec 1 18:00:00 2019`, the parsing function can be defined as:

```
bsub_opt$parse_time = function(x) {
  as.POSIXct(x, format = "\\%b \\%d \\%H:\\%M:\\%S \\%Y")
}
```

**Value**

The corresponding option values.

**Examples**

```
# The default bsub_opt
bsub_opt
```

---

config_bsub	<i>Configure bsub global options</i>
-------------	--------------------------------------

---

**Description**

Configure bsub global options

**Usage**

```
config_bsub(verbose = TRUE)
```

**Arguments**

verbose	Whether to print messages.
---------	----------------------------

**Details**

It sets the submission nodes, user name and how to call Rscript.

---

formatFileSize	<i>Format file size</i>
----------------	-------------------------

---

**Description**

Format file size

**Usage**

```
formatFileSize(table, columns)
```

**Arguments**

table	Internally used.
columns	Internally used.

---

job\_dependency\_all      *Job dependencies*

---

### Description

Job dependencies

### Usage

```
job_dependency_all(job_tb = NULL)

job_dependency_igraph(job_id, job_tb = NULL)

job_dependency_dot(job_id, job_tb = NULL, use_label = FALSE, label_width = 15)

job_dependency_diagram(
  job_id,
  job_tb = NULL,
  use_label = FALSE,
  label_width = 15,
  ...
)
```

### Arguments

job_tb	A data frame from <code>bjobs()</code> . Internally used.
job_id	A single job ID.
use_label	Whether to use job names on the diagram?
label_width	Max number of characters for wrapping the label into lines.
...	Pass to <code>DiagrammeR::grViz()</code> , such as the size of the html widget.

### Value

`job_dependency_all()` returns a list that contains three elements:

- `dep_mat`: a two column matrix containing dependencies from parents to children.
- `id2name`: a named vector containing mapping from job IDs to job names.
- `id2stat`: a named vector containing mapping from job IDs to job status.

`job_dependency_igraph()` returns a `igraph::igraph` object which contains a dependency graph induced by the input job ID.

`job_dependency_dot()` returns a DOT code for GraphViz visualization.

`job_dependency_diagram()` makes a HTML-based dependency diagram.

**Examples**

```
## Not run:
job1 = random_job()
job2 = random_job()
job3 = random_job(dependency = c(job1, job2))

job_dependency_all()
job_dependency_igraph(job3)
cat(job_dependency_dot(job3))
job_dependency_diagram(job3)

## End(Not run)
```

---

 job\_log

---

*Obtain Job log*


---

**Description**

Obtain Job log

**Usage**

```
job_log(job_id, print = TRUE, n_line = 10)
```

**Arguments**

job_id	The job id. It can be a single job or a vector of job ids.
print	Whether to print the log message to the terminal.
n_line	Number of last lines for each job to show when multiple jobs are queried.

**Value**

The log messages as a vector.

**Examples**

```
## Not run:
# a single job
job_id = 1234567 # job ids can be get from `bjobs`
job_log(job_id)
# multiple jobs
job_id = c(10000000, 10000001, 10000002)
job_log(job_id) # by default last 10 lines for each job are printed
job_log(job_id, n_line = 20) # print last 20 lines for each job
# logs for all running jobs
job_log()

## End(Not run)
```

---

 job\_rerun

*Rerun jobs*


---

**Description**

Rerun jobs

**Usage**

```
job_rerun(job_id, dependency = character(), verbose = TRUE, job_tb = NULL)
```

```
pipeline_rerun(job_id, skip_done = TRUE, verbose = TRUE)
```

**Arguments**

job_id	A single job ID. In pipeline_rerun(), it is the job ID of any one job in the pipeline.
dependency	A vector of job IDs that current job depends on.
verbose	Whether to print messages.
job_tb	The data frame returned from <a href="#">bjobs()</a> , internally used.
skip_done	Whether to skip done jobs.

**Details**

In pipeline\_rerun(), the full set of jobs can be captured by one job in the pipeline.

**Value**

job\_rerun() returns the job IDs. pipeline\_rerun() returns NULL.

---

 job\_status\_by\_name

*Job status by job ID or name*


---

**Description**

Job status by job ID or name

**Usage**

```
job_status_by_name(job_name)
```

```
job_status_by_id(job_id)
```

**Arguments**

job\_name        A single job name.  
job\_id         A single job ID.

**Value**

A vector of job status, with job IDs are names.

**Examples**

```
## Not run:  
job_status_by_name("example")  
job_status_by_id(123456)  
  
## End(Not run)
```

---

list_dump_files	<i>Check whether there are dump files</i>
-----------------	---

---

**Description**

Check whether there are dump files

**Usage**

```
list_dump_files(print = TRUE)
```

**Arguments**

print            Whether to print messages.

**Details**

For the failed jobs, LSF cluster might generate a core dump file and R might generate a .RDataTmp file.

Note if you manually set working directory in your R code/script, the R dump file can be not caught.

**Value**

A vector of file names.

**Examples**

```
## Not run:  
list_dump_files()  
  
## End(Not run)
```

---

list_temp_files	<i>Clear temporary dir</i>
-----------------	----------------------------

---

**Description**

Clear temporary dir

**Usage**

```
list_temp_files()
remove_temp_files(ask = TRUE)
```

**Arguments**

ask                    Whether to promote.

**Details**

The temporary files might be used by the running/pending jobs. Deleting them would affect some jobs. You should better delete them after all jobs are done.

**Examples**

```
## Not run:
list_temp_files()

## End(Not run)
```

---

monitor	<i>Browser-based interactive job monitor</i>
---------	--

---

**Description**

Browser-based interactive job monitor

**Usage**

```
monitor()
```

**Details**

The monitor is implemented as a shiny app.

**Examples**

```
## Not run:
monitor()

## End(Not run)
```

---

print.bjobs	<i>Summary of jobs</i>
-------------	------------------------

---

**Description**

Summary of jobs

**Usage**

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

**Arguments**

x	A bjobs object.
...	other arguments.

---

random_job	<i>Submit a random job</i>
------------	----------------------------

---

**Description**

Submit a random job

**Usage**

```
random_job(name, secs = 30, ...)
```

**Arguments**

name	Job name.
secs	Seconds to sleep.
...	Pass to <a href="#">bsub_chunk()</a> .

**Details**

It simply runs `Sys.sleep(secs)` in the job.

**Value**

A job ID.

**Examples**

```
## Not run:
random_job()

## End(Not run)
```

---

retrieve_var	<i>Retrieve saved variable</i>
--------------	--------------------------------

---

**Description**

Retrieve saved variable

**Usage**

```
retrieve_var(job_id, job_name = NULL, wait = 30)
```

**Arguments**

job_id	A single job ID.
job_name	A single job name. Since jobs may have the same names, the most recent job is selected.
wait	Seconds to wait until the job is finished.

**Details**

It retrieves the saved variable in [bsub\\_chunk\(\)](#) when `save_rds = TRUE` is set.

**Value**

The retrieved object.

**Examples**

```
## Not run:
job_id = bsub_chunk(name = "example", save_var = TRUE,
{
  Sys.sleep(10)
  1+1
})
retrieve_var(job_id)

## End(Not run)
```

---

run_cmd	<i>Run command on submission node</i>
---------	---------------------------------------

---

**Description**

Run command on submission node

**Usage**

```
run_cmd(cmd, print = FALSE)
```

**Arguments**

cmd	A single-line command.
print	Whether to print output from the command.

**Details**

If current node is not the submission node, the command is executed via ssh.

**Value**

The output of the command.

**Examples**

```
## Not run:  
# run pwd on remote node  
run_cmd("pwd")  
  
## End(Not run)
```

---

ssh_connect	<i>Connect to submission via ssh</i>
-------------	--------------------------------------

---

**Description**

Connect to submission via ssh

**Usage**

```
ssh_connect(...)  
  
ssh_disconnect()
```

**Arguments**

... Pass to `ssh::ssh_connect()`.

**Examples**

```
# ssh is automatically connected. To manually connect ssh, run:  
## Not run:  
ssh_connect()  
ssh_disconnect()  
  
## End(Not run)  
# where the user name is the one you set in `bsub_opt$user` and  
# the node is the one you set in `bsub_opt$login_node`.
```

# Index

- \* **datasets**
  - bconf, 2
  - .GlobalEnv, 8
- bconf, 2
- bjobs, 3
- bjobs(), 6, 13, 15
- bjobs\_barplot, 5
- bjobs\_done (bjobs), 3
- bjobs\_exit (bjobs), 3
- bjobs\_pending (bjobs), 3
- bjobs\_raw (bjobs), 3
- bjobs\_reset\_timestamp, 6
- bjobs\_running (bjobs), 3
- bjobs\_timeline (bjobs\_barplot), 5
- bkill, 6
- bpercent (bjobs), 3
- bsub\_chunk, 7
- bsub\_chunk(), 18, 19
- bsub\_cmd (bsub\_chunk), 7
- bsub\_opt, 10
- bsub\_opt(), 3
- bsub\_script (bsub\_chunk), 7

config\_bsub, 12

DiagrammeR::grViz(), 13

digest::digest(), 8

formatFileSize, 12

igraph::igraph, 13

job\_dependency\_all, 13

job\_dependency\_diagram  
(job\_dependency\_all), 13

job\_dependency\_dot  
(job\_dependency\_all), 13

job\_dependency\_igraph  
(job\_dependency\_all), 13

job\_log, 14

job\_rerun, 15

job\_status\_by\_id (job\_status\_by\_name),  
15

job\_status\_by\_name, 15

list\_dump\_files, 16

list\_temp\_files, 17

monitor, 17

pipeline\_rerun (job\_rerun), 15

print.bconf (bconf), 2

print.bjobs, 18

random\_job, 18

remove\_temp\_files (list\_temp\_files), 17

retrieve\_var, 19

retrieve\_var(), 9

run\_cmd, 20

ssh::ssh\_connect(), 21

ssh\_connect, 20

ssh\_disconnect (ssh\_connect), 20