

# Beanstalk Protocol

---

## Protocol

The beanstalk protocol runs over TCP using ASCII encoding. Clients connect, send commands and data, wait for responses, and close the connection. For each connection, the server processes commands serially in the order in which they were received and sends responses in the same order. All integers in the protocol are formatted in decimal and (unless otherwise indicated) non-negative.

Names, in this protocol, are ASCII strings. They may contain letters (A-Z and a-z), numerals (0-9), hyphen ("-"), plus ("+"), slash ("/"), semicolon (";"), dot ("."), dollar-sign ("\$"), underscore ("\_"), and parentheses ("(" and ")"), but they may not begin with a hyphen. They are terminated by white space (either a space char or end of line). Each name must be at least one character long.

The protocol contains two kinds of data: text lines and unstructured chunks of data. Text lines are used for client commands and server responses. Chunks are used to transfer job bodies and stats information. Each job body is an opaque sequence of bytes. The server never inspects or modifies a job body and always sends it back in its original form. It is up to the clients to agree on a meaningful interpretation of job bodies.

The client may issue the **quit** command, or simply close the TCP connection when it no longer has use for the server. However, **beanstalkd** performs very well with a large number of open connections, so it is usually better for the client to keep its connection open and reuse it as much as possible. This also avoids the overhead of establishing new TCP connections.

If a client violates the protocol (such as by sending a request that is not well-formed or a command that does not exist) or if the server has an error, the server will reply with one of the following error messages:

### **OUT\_OF\_MEMORY\r\n**

The server cannot allocate enough memory for the job. The client should try again later.

### **INTERNAL\_ERROR\r\n**

This indicates a bug in the server. It should never happen. If it does happen, please report it at <http://groups.google.com/group/beanstalk-talk>.

### **BAD\_FORMAT\r\n**

The client sent a command line that was not well-formed. This can happen if the line does not end with "\r\n", if non-numeric characters occur where an integer is expected, if the wrong number of arguments is present, or if the command line is mal-formed in any other way.

### **UNKNOWN\_COMMAND\r\n**

The client sent a command that the server does not know.

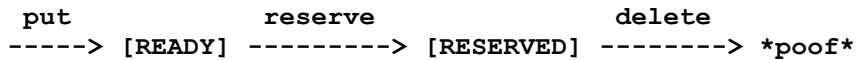
These error responses will not be listed in this document for individual commands in the following sections, but they are implicitly included in the description of all commands. Clients should be prepared to receive an error response after any command.

As a last resort, if the server has a serious error that prevents it from continuing service to the current client, the server will close the connection.

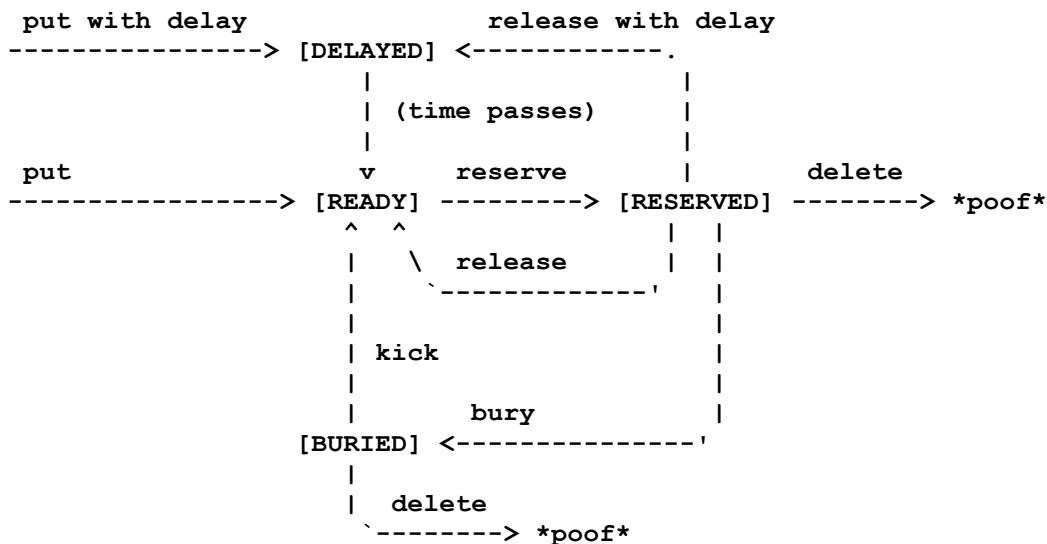
# Job Lifecycle

A job in beanstalk gets created by a client with the **put** command. During its life it can be in one of four states: **ready**, **reserved**, **delayed**, or **buried**. After the put command, a job typically starts out ready. It waits in the ready queue until a worker comes along and runs the **reserve** command. If this job is next in the queue, it will be reserved for the worker. The worker will execute the job; when it is finished the worker will send a **delete** command to delete the job.

Here is a picture of the typical job lifecycle:



Here is a picture with more possibilities:



The system has one or more tubes. Each tube consists of a ready queue and a delay queue. Each job spends its entire life in one tube. Consumers can show interest in tubes by sending the **watch** command; they can show disinterest by sending the **ignore** command. This set of interesting tubes is said to be a consumer's "watch list". When a client reserves a job, it may come from any of the tubes in its watch list.

When a client connects, its watch list is initially just the tube named **default**. If it submits jobs without having sent a **use** command, they will live in the tube named **default**.

Tubes are created on demand whenever they are referenced. If a tube is empty (that is, it contains no ready, delayed, or buried jobs) and no client refers to it, it will be deleted.

## Producer Commands

The **put** command is for any process that wants to insert a job into the queue. It comprises a command line followed by the job body:

```
put <pri> <delay> <ttr> <bytes>\r\n
<data>\r\n
```

It inserts a job into the client's currently used tube (see the **use** command below).

- **<pri>** is an integer  $< 2^{32}$ . Jobs with smaller priority values will be scheduled before jobs with larger priorities. The most urgent priority is 0; the least urgent priority is 4,294,967,295.
- **<delay>** is an integer number of seconds to wait before putting the job in the ready queue. The job will be in the **delayed** state during this time.
- **<ttr>** -- time to run -- is an integer number of seconds to allow a worker to run this job. This time is counted from the moment a worker reserves this job. If the worker does not delete, release, or bury the job within **<ttr>** seconds, the job will time out and the server will release the job. The minimum ttr is 1. If the client sends 0, the server will silently increase the ttr to 1.
- **<bytes>** is an integer indicating the size of the job body, not including the trailing "\r\n". This value must be less than max-job-size (default:  $2^{16}$ ).
- **<data>** is the job body -- a sequence of bytes of length **<bytes>** from the previous line.

After sending the command line and body, the client waits for a reply, which may be:

**INSERTED <id>\r\n**  
to indicate success.

- **<id>** is the integer id of the new job

**BURIED <id>\r\n**  
if the server ran out of memory trying to grow the priority queue data structure.

- **<id>** is the integer id of the new job

**EXPECTED\_CRLF\r\n**  
The job body must be followed by a CR-LF pair, that is, "\r\n". These two bytes are not counted in the job size given by the client in the put command line.

**JOB\_TOO\_BIG\r\n**  
The client has requested to put a job with a body larger than max-job-size bytes.

**DRAINING\r\n**  
This means that the server has been put into "drain mode" and is no longer accepting new jobs. The client should try another server or disconnect and try again later.

The **use** command is for producers. Subsequent put commands will put jobs into the tube specified by this command. If no use command has been issued, jobs will be put into the tube named **default**.

**use <tube>\r\n**

- **<tube>** is a name at most 200 bytes. It specifies the tube to use. If the tube does not exist, it will be created.

The only reply is:

**USING <tube>\r\n**

- **<tube>** is the name of the tube now being used.

## Worker Commands

A process that wants to consume jobs from the queue uses **reserve**, **delete**, **release**, and **bury**. The first worker command, **reserve**, looks like this:

```
reserve\r\n
```

Alternatively, you can specify a timeout as follows:

```
reserve-with-timeout <seconds>\r\n
```

This will return a newly-reserved job. If no job is available to be reserved, **beanstalkd** will wait to send a response until one becomes available. Once a job is reserved for the client, the client has limited time to run (TTR) the job before the job times out. When the job times out, the server will put the job back into the ready queue. Both the TTR and the actual time left can be found in response to the stats-job command.

If more than one job is ready, **beanstalkd** will choose the one with the smallest priority value. Within each priority, it will choose the one that was received first.

A timeout value of 0 will cause the server to immediately return either a response or **TIMED\_OUT**. A positive value of timeout will limit the amount of time the client will block on the reserve request until a job becomes available.

During the TTR of a reserved job, the last second is kept by the server as a safety margin, during which the client will not be made to wait for another job. If the client issues a reserve command during the safety margin, or if the safety margin arrives while the client is waiting on a reserve command, the server will respond with:

```
DEADLINE_SOON\r\n
```

This gives the client a chance to delete or release its reserved job before the server automatically releases it.

```
TIMED_OUT\r\n
```

If a non-negative timeout was specified and the timeout exceeded before a job became available, or if the client's connection is half-closed, the server will respond with **TIMED\_OUT**.

Otherwise, the only other response to this command is a successful reservation in the form of a text line followed by the job body:

```
RESERVED <id> <bytes>\r\n  
<data>\r\n
```

- **<id>** is the job id -- an integer unique to this job in this instance of beanstalkd.
- **<bytes>** is an integer indicating the size of the job body, not including the trailing "**\r\n**".
- **<data>** is the job body -- a sequence of bytes of length **<bytes>** from the previous line. This is a verbatim copy of the bytes that were originally sent to the server in the put command for this job.

The delete command removes a job from the server entirely. It is normally used by the client when the job has successfully run to completion. A client can delete jobs that it has reserved, ready jobs, delayed jobs, and jobs that are buried. The delete command looks like this:

```
delete <id>\r\n
```

- **<id>** is the job id to delete.

The client then waits for one line of response, which may be:

**DELETED\r\n**

to indicate success.

**NOT\_FOUND\r\n**

if the job does not exist or is not either reserved by the client, ready, or buried. This could happen if the job timed out before the client sent the delete command.

The release command puts a reserved job back into the ready queue (and marks its state as *ready*) to be run by any client. It is normally used when the job fails because of a transitory error. It looks like this:

**release <id> <pri> <delay>\r\n**

- **<id>** is the job id to release.
- **<pri>** is a new priority to assign to the job.
- **<delay>** is an integer number of seconds to wait before putting the job in the ready queue. The job will be in the *delayed* state during this time.

The client expects one line of response, which may be:

**RELEASED\r\n**

to indicate success.

**BURIED\r\n**

if the server ran out of memory trying to grow the priority queue data structure.

**NOT\_FOUND\r\n**

if the job does not exist or is not reserved by the client.

The bury command puts a job into the *buried* state. Buried jobs are put into a FIFO linked list and will not be touched by the server again until a client kicks them with the *kick* command.

The bury command looks like this:

**bury <id> <pri>\r\n**

- **<id>** is the job id to release.
- **<pri>** is a new priority to assign to the job.

There are two possible responses:

**BURIED\r\n**

to indicate success.

**NOT\_FOUND\r\n**

if the job does not exist or is not reserved by the client.

The *touch* command allows a worker to request more time to work on a job. This is useful for jobs that potentially take a long time, but you still want the benefits of a TTR pulling a job away from an unresponsive worker. A worker may periodically tell the server that it's still alive and processing a job (e.g. it may do this on

**DEADLINE\_SOON**). The command postpones the auto release of a reserved job until **TTR** seconds from when the command is issued.

The touch command looks like this:

```
touch <id>\r\n
```

- **<id>** is the ID of a job reserved by the current connection.

There are two possible responses:

```
TOUCHED\r\n
```

to indicate success.

```
NOT_FOUND\r\n
```

if the job does not exist or is not reserved by the client.

The **watch** command adds the named tube to the watch list for the current connection. A reserve command will take a job from any of the tubes in the watch list. For each new connection, the watch list initially consists of one tube, named **default**.

```
watch <tube>\r\n
```

- **<tube>** is a name at most 200 bytes. It specifies a tube to add to the watch list. If the tube doesn't exist, it will be created.

The reply is:

```
WATCHING <count>\r\n
```

- **<count>** is the integer number of tubes currently in the watch list.

The **ignore** command is for consumers. It removes the named tube from the watch list for the current connection.

```
ignore <tube>\r\n
```

The reply is one of:

```
WATCHING <count>\r\n
```

to indicate success.

- **<count>** is the integer number of tubes currently in the watch list.

```
NOT_IGNORED\r\n
```

if the client attempts to ignore the only tube in its watch list.

## Other Commands

The peek commands let the client inspect a job in the system. There are four variations. All but the first operate only on the currently used tube.

```
peek <id>\r\n
```

return job **<id>**.

**peek-ready\r\n**

return the next ready job.

**peek-delayed\r\n**

return the delayed job with the shortest delay left.

**peek-buried\r\n**

return the next job in the list of buried jobs.

There are two possible responses, either a single line:

**NOT\_FOUND\r\n**

if the requested job doesn't exist or there are no jobs in the requested state.

Or a line followed by a chunk of data, if the command was successful:

**FOUND <id> <bytes>\r\n**

**<data>\r\n**

- **<id>** is the job id.
- **<bytes>** is an integer indicating the size of the job body, not including the trailing "**\r\n**".
- **<data>** is the job body -- a sequence of bytes of length **<bytes>** from the previous line.

The kick command applies only to the currently used tube. It moves jobs into the ready queue. If there are any buried jobs, it will only kick buried jobs. Otherwise it will kick delayed jobs. It looks like:

**kick <bound>\r\n**

- **<bound>** is an integer upper bound on the number of jobs to kick. The server will kick no more than **<bound>** jobs.

The response is of the form:

**KICKED <count>\r\n**

- **<count>** is an integer indicating the number of jobs actually kicked.

The kick-job command is a variant of kick that operates with a single job identified by its job id. If the given job id exists and is in a buried or delayed state, it will be moved to the ready queue of the the same tube where it currently belongs. The syntax is:

**kick-job <id>\r\n**

- **<id>** is the job id to kick.

The response is one of:

**NOT\_FOUND\r\n**

if the job does not exist or is not in a kickable state. This can also happen upon internal errors.

**KICKED\r\n**

when the operation succeeded.

The stats-job command gives statistical information about the specified job if it exists. Its form is:

**stats-job <id>\r\n**

- **<id>** is a job id.

The response is one of:

**NOT\_FOUND\r\n**

if the job does not exist.

**OK <bytes>\r\n<data>\r\n**

- **<bytes>** is the size of the following data section in bytes.
- **<data>** is a sequence of bytes of length **<bytes>** from the previous line. It is a **YAML** file with statistical information represented a dictionary.

The stats-job data is a **YAML** file representing a single dictionary of strings to scalars. It contains these keys:

- **id** is the job id
- **tube** is the name of the tube that contains this job
- **state** is **ready** or **delayed** or **reserved** or **buried**
- **pri** is the priority value set by the put, release, or bury commands.
- **age** is the time in seconds since the put command that created this job.
- **time-left** is the number of seconds left until the server puts this job into the ready queue. This number is only meaningful if the job is reserved or delayed. If the job is reserved and this amount of time elapses before its state changes, it is considered to have timed out.
- **file** is the number of the earliest binlog file containing this job. If -b wasn't used, this will be 0.
- **reserves** is the number of times this job has been reserved.
- **timeouts** is the number of times this job has timed out during a reservation.
- **releases** is the number of times a client has released this job from a reservation.
- **buries** is the number of times this job has been buried.
- **kicks** is the number of times this job has been kicked.

The stats-tube command gives statistical information about the specified tube if it exists. Its form is:

**stats-tube <tube>\r\n**

- **<tube>** is a name at most 200 bytes. Stats will be returned for this tube.

The response is one of:

**NOT\_FOUND\r\n**

if the tube does not exist.

**OK <bytes>\r\n<data>\r\n**

- **<bytes>** is the size of the following data section in bytes.



- **<data>** is a sequence of bytes of length **<bytes>** from the previous line. It is a **YAML** file with statistical information represented a dictionary.

The stats-tube data is a **YAML** file representing a single dictionary of strings to scalars. It contains these keys:

- **name** is the tube's name.
- **current-jobs-urgent** is the number of ready jobs with priority < 1024 in this tube.
- **current-jobs-ready** is the number of jobs in the ready queue in this tube.
- **current-jobs-reserved** is the number of jobs reserved by all clients in this tube.
- **current-jobs-delayed** is the number of delayed jobs in this tube.
- **current-jobs-buried** is the number of buried jobs in this tube.
- **total-jobs** is the cumulative count of jobs created in this tube in the current **beanstalkd** process.
- **current-using** is the number of open connections that are currently using this tube.
- **current-waiting** is the number of open connections that have issued a reserve command while watching this tube but not yet received a response.
- **current-watching** is the number of open connections that are currently watching this tube.
- **pause** is the number of seconds the tube has been paused for.
- **cmd-delete** is the cumulative number of delete commands for this tube
- **cmd-pause-tube** is the cumulative number of pause-tube commands for this tube.
- **pause-time-left** is the number of seconds until the tube is un-paused.

The stats command gives statistical information about the system as a whole. Its form is:

```
stats\r\n
```

The server will respond:

```
OK <bytes>\r\n
<data>\r\n
```

- **<bytes>** is the size of the following data section in bytes.
- **<data>** is a sequence of bytes of length **<bytes>** from the previous line. It is a **YAML** file with statistical information represented a dictionary.

The stats data for the system is a **YAML** file representing a single dictionary of strings to scalars. Entries described as **cumulative** are reset when the **beanstalkd** process starts; they are not stored on disk with the **-b** flag.

- **current-jobs-urgent** is the number of ready jobs with priority < 1024.
- **current-jobs-ready** is the number of jobs in the ready queue.
- **current-jobs-reserved** is the number of jobs reserved by all clients.
- **current-jobs-delayed** is the number of delayed jobs.
- **current-jobs-buried** is the number of buried jobs.
- **cmd-put** is the cumulative number of put commands.
- **cmd-peek** is the cumulative number of peek commands.
- **cmd-peek-ready** is the cumulative number of peek-ready commands.
- **cmd-peek-delayed** is the cumulative number of peek-delayed commands.
- **cmd-peek-buried** is the cumulative number of peek-buried commands.
- **cmd-reserve** is the cumulative number of reserve commands.
- **cmd-use** is the cumulative number of use commands.
- **cmd-watch** is the cumulative number of watch commands.
- **cmd-ignore** is the cumulative number of ignore commands.

- **cmd-delete** is the cumulative number of delete commands.
- **cmd-release** is the cumulative number of release commands.
- **cmd-bury** is the cumulative number of bury commands.
- **cmd-kick** is the cumulative number of kick commands.
- **cmd-stats** is the cumulative number of stats commands.
- **cmd-stats-job** is the cumulative number of stats-job commands.
- **cmd-stats-tube** is the cumulative number of stats-tube commands.
- **cmd-list-tubes** is the cumulative number of list-tubes commands.
- **cmd-list-tube-used** is the cumulative number of list-tube-used commands.
- **cmd-list-tubes-watched** is the cumulative number of list-tubes-watched commands.
- **cmd-pause-tube** is the cumulative number of pause-tube commands.
- **job-timeouts** is the cumulative count of times a job has timed out.
- **total-jobs** is the cumulative count of jobs created.
- **max-job-size** is the maximum number of bytes in a job.
- **current-tubes** is the number of currently-existing tubes.
- **current-connections** is the number of currently open connections.
- **current-producers** is the number of open connections that have each issued at least one put command.
- **current-workers** is the number of open connections that have each issued at least one reserve command.
- **current-waiting** is the number of open connections that have issued a reserve command but not yet received a response.
- **total-connections** is the cumulative count of connections.
- **pid** is the process id of the server.
- **version** is the version string of the server.
- **rusage-utime** is the cumulative user CPU time of this process in seconds and microseconds.
- **rusage-stime** is the cumulative system CPU time of this process in seconds and microseconds.
- **uptime** is the number of seconds since this server process started running.
- **binlog-oldest-index** is the index of the oldest binlog file needed to store the current jobs.
- **binlog-current-index** is the index of the current binlog file being written to. If binlog is not active this value will be 0.
- **binlog-max-size** is the maximum size in bytes a binlog file is allowed to get before a new binlog file is opened.
- **binlog-records-written** is the cumulative number of records written to the binlog.
- **binlog-records-migrated** is the cumulative number of records written as part of compaction.
- **id** is a random id string for this server process, generated when each **beanstalkd** process starts.
- **hostname** the hostname of the machine as determined by `uname`.

The list-tubes command returns a list of all existing tubes. Its form is:

```
list-tubes\r\n
```

The response is:

```
OK <bytes>\r\n
<data>\r\n
```

- **<bytes>** is the size of the following data section in bytes.
- **<data>** is a sequence of bytes of length **<bytes>** from the previous line. It is a **YAML** file containing all tube names as a list of strings.

The list-tube-used command returns the tube currently being used by the client. Its form is:

```
list-tube-used\r\n
```

The response is:

**USING <tube>\r\n**

- **<tube>** is the name of the tube being used.

The list-tubes-watched command returns a list tubes currently being watched by the client. Its form is:

**list-tubes-watched\r\n**

The response is:

**OK <bytes>\r\n  
<data>\r\n**

- **<bytes>** is the size of the following data section in bytes.
- **<data>** is a sequence of bytes of length **<bytes>** from the previous line. It is a **YAML** file containing watched tube names as a list of strings.

The quit command simply closes the connection. Its form is:

**quit\r\n**

The pause-tube command can delay any new job being reserved for a given time. Its form is:

**pause-tube <tube-name> <delay>\r\n**

- **<tube>** is the tube to pause
- **<delay>** is an integer number of seconds to wait before reserving any more jobs from the queue

There are two possible responses:

**PAUSED\r\n**  
to indicate success.

**NOT\_FOUND\r\n**  
if the tube does not exist.