cmlkeep: Process Keep Alive Version 1 Reference and User Guide

CML00092-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Introduction

Code Magus Keep Alive is a suite of programs used for starting and managing critical processes. Each invocation of a process (or target process) is called a job and is referred to by a unique job name. Keep Alive can be used to start a job and ensure that it is always running by restarting it if it terminates. It also allows a user to monitor the status of all jobs and to send a signal to any one of them.

1.1 Keep Alive

The Code Magus tool `cmlkeep` will run and continuously restart a target process specified in the command line options. This means that any server or daemon that is always required to be running is immediately restarted should it terminate either through a fault (an internal error or an external agent such as the Linux out-of-memory killer) or by intention. If there is a time span before which it is inadvisable to restart a process a sleep time can be specified in seconds which will cause `cmlkeep` to wait before restarting the target process. Often TCP/IP ports require up to 4 minutes while they wait for any lost message segments in the network to expire before they can be re-used again.

Each job must have a unique name as known to `cmlkeep`; This is either specified as a command line option or is the base name of the target process. Additionally a title may be specified for this job for reporting purposes or as an aid to diagnosis should the need arise.

There is the facility to specify the maximum number of restarts allowed within a time window so that a forever failing target process will not be restarted ad infinitum.

`cmlkeep` can also capture the standard output and standard error streams of the target process and write them to a variable length RDW based log. An RDW log is a record based file where the length of each record precedes the record data and is readable using the Binary Access Method[2]. The log file name is generated from a mask and is based on a sequence and / or a time stamp. The RDW log file may be automatically managed in that the file is written out in segments when some threshold is reached or on receiving a specified signal.

If a target process is restarted there is the option to run a notify process in order to notify a person or process outside of the normal log output of `cmlkeep`. The notify process may be an executable or a shell script and is started with its standard input connected to a pipe to which `cmlkeep` writes a message detailing the fault followed by the contents of a file if specified as a start up option on the command line. Examples of such a notify process would be to email, text or page support personnel.

Under some circumstances a target process may be deemed to complete normally. A normal completion of a target process may require a restart or a halt to the restart cycle of the target process. There are command line options to allow any combination of

halt or restart by specifying the return code or termination signal produced by the target process. If the conditions require a restart the notification process is not executed and the target process is silently restarted. If it requires a halt of the restart cycle then `cmlkeep` will gracefully end and not restart the target process any more.

1.2 Monitoring jobs

The Code Magus tool `cmljobs` allows a user to check any job running under the control of `cmlkeep`. It can also be used for administration of the keep system. For example in the event that both `cmlkeep` and the target process are terminated by an external agent, `cmljobs` can be used to clean up or remove the reference to the job.

1.3 Signalling jobs

The Code Magus tool `cmlkill` operates in a similar way to the Linux utility `kill` in that it can send a signal to a target process. The job is named by its name rather than having to use the Process Identifier (PID).

2 Job Keep alive Processing

2.1 Introduction

The Code Magus Keep Alive program `cmlkeep` is a way to run a target process referred to by a job name rather than a PID. Specifically its main purpose is to make sure that the target process is always running. This is important for a number of reasons such as

- Restarting a target process if it terminates abnormally.
- Restarting a target process if an out side agent terminates the target process with a signal. On Linux this could happen if the overall memory usage exceeds a set limit causing the 'Out of Memory Killer' (OOM Killer) to kill (abnormally terminate) any process in order to regain the memory.
- A target process needs to be recycled in order for it to restart with an updated configuration.
- A target process needs to be gracefully shut down, possibly for a machine reboot.

An instance of a `cmlkeep` job can be configured such that all of the above points can be supported.

2.2 Command line parameters

If `cmlkeep` is started with the `--help` option it will only print help about the command line options with a short description of each as follows.

```
Usage: cmlkeep [OPTIONS]* -- <Target-Process-program> [args]*
          If there are no OPTIONS then -- can be omitted.
          For try-harder-to-restart: Attempt this many extra
          restart windows, with an incremented delay between
          successive ones (increment is --restart-window)
          Note: -1 means forever

  -j, --job-name=<string>                          Job name (must be unique)
  -t, --title=<string>                             Title of this keep alive
                                                   process
  -s, --sleep={10|<seconds>}                       Sleep time before restarting
                                                   the Target-Process
  -n, --notify-command=<command>                   Notify command to run when
                                                   Target-Process ends
  -f, --notify-file=<text-file>                    File of text passed as
                                                   standard in to the notify
                                                   command
  -m, --restart-max={5|<integer>}                  Maximum times allowed to
                                                   restart in restart window
  -w, --restart-window={60|<seconds>}              Restart window length
```

```
  -y, --try-harder-to-restart={0|-1|<attempts>}     Try harder to restart.
  -e, --normal-exit-rc=<integer>                    Exit if Target-Process ends
                                                    with this return code
  -r, --normal-restart-rc=<integer>                 Restart Target-Process, with
                                                    no notification, if it
                                                    ends with this return code
  -E, --normal-exit-signal=<integer>                Exit if Target-Process
                                                    ends with this signal
  -R, --normal-restart-signal=<integer>             Restart Target-Process, with
                                                    no notification, if it
                                                    ends with this signal
  -H, --hup                                         Do not ignore SIGHUP
  -k, --keep-log-spec={stderr|<am>(<obj>[,<opts>])} AM open string for cmlkeep
                                                    progress log
  -l, --log-file={<filename>|<pattern>}             Pattern for log file name
                                                    and path to which to log
                                                    Target-process standard
                                                    output and error to
  -T, --log-name-strftime                           Use strftime on log name
                                                    pattern string
  -B, --log-segment-bytes=<byte-count>              Log segments switched after
                                                    byte count reached
  -S, --log-segment-seconds=<seconds>               Log segments switched after
                                                    time in seconds expires
  -C, --log-segment-records=<records>               Log segments switched after
                                                    record count reached
  -I, --log-segment-signal={0|<signum>}             Log segments switched on
                                                    reciept of signal
  -F, --log-flush-writes                            Log stream to be flushed
                                                    after each write
  -O, --do-not-capture-output                       Do not capture standard
                                                    output and error from the
                                                    Target-Process
  -J, --print-job-directory                         Print the job directory only
  -v, --verbose                                     Verbose processing mode

Help options:
  -?, --help                                        Show this help message
      --usage                                       Display brief usage message
```

These parameters or options are further explained in the following sections.

2.2.1 -j, –job-name

`-j, --job-name=<string>`

This option is the job name of the target process invoked by `cmlkeep`. It must be unique across all jobs that are invoked by `cmlkeep`. It is possible to configure a `cmlkeep` environment such that it uses more than one job base, but is not recommended. This job name is used by both `cmljobs` and `cmlkill` to refer to the job

running this target process.

2.2.2 -t, –title

```
-t, --title=<string>
```

This option specifies the title of the job and is useful when reporting on jobs or diagnosing problems to do with it.

2.2.3 -s, –sleep

```
-s, --sleep={10|<seconds>}
```

This option specifies the time in seconds that `cmlkeep` will wait (or sleep) before restarting the target process.

2.2.4 -n, –notify-command

```
-n, --notify-command=<command>
```

This option specifies the name of a notify process that will be executed when a target process is restarted due to an abnormal termination. A message detailing the reason for the restart is made available to the notify process through its standard input stream.

Generally the notify process reads from its standard input and notifies support via some mechanism. A good example on Linux is to use `mutt` to email the message to a number of recipients. The command would look like this

```
--notify-command="mutt -s \ 'Error: Application MyApp on \
   machine.domain.com has restarted' support@company.com"
```

2.2.5 -f, –notify-file

```
-f, --notify-file=<text-file>
```

This option names a file of text that is made available to the notify process on its standard input stream after the generated message. This text is the same for every restart and could include company or support contact details or support action notes for the application that has been restarted.

2.2.6 -m, –restart-max

`-m, --restart-max={5|<integer>}`

This option specifies the maximum number of times a target process can be restarted in the `--restart-window`. If this number is exceeded the restart cycle is halted and the notify message will be changed to indicate this.

2.2.7 -w, –restart-window

`-w, --restart-window={60|<seconds>}`

This option specifies in seconds the span of the restart window. Note that the time specified in `--sleep` needs to be factored in to the time it takes for a target process to restart and terminate.

2.2.8 -y, –try-harder-to-restart

`-y, --try-harder-to-restart={0|-1|<attempts>}`

This option is only used in the scenario where a target process fails immediately and repeatedly more than `--restart-max` times within `--restart-window` seconds.

There are three ways that this option can be used. They are

1. Specify a value of zero (the default)

   This is the normal or default value for this option. When used and this restart scenario occurs `cmlkeep` will not attempt another restart and will terminate.

2. Specify a positive integer.

   If this option is set to a value greater than zero then `cmlkeep` will attempt this many extra successive restart windows in an attempt to successfully restart the target process.

   It will wait `--restart-window` seconds more between each restart window in the hope that this will allow a sysadmin to rectify the situation that is causing the target process not to start. In other words the rate of restart is slowed by `--restart-window` seconds more as successive restart windows are attempted. If the target process starts successfully then this history of successive restart windows is reset back to the original value.

   A usual cause is for example when a file system fills up and a target process can not write an output file and therefore ends; on restart by `cmlkeep` it again can not write the log file and again terminates. This then starts the cycle of immediate

restarts within a restart window and possibly successive restart windows. Once a sysadmin has rectified the situation (made space available) `cmlkeep` will successfully restart the target process and the restart window history is discarded and reset to the initial value.

3. Specify any negative integer.

   If this option is set to a negative integer then `cmlkeep` will continuously attempt to restart the target process.

   It still uses the concept of the restart window time, but the wait between each window is initially set to `--restart-window` and only increased by 1 second each time.

2.2.9 -e, –normal-exit-rc

`-e, --normal-exit-rc=<integer>`

This option specifies an integer and if the target process ends with a return code equal to this the restart cycle will be halted and `cmlkeep` will gracefully end. If this option is not specified then the target process will always be restarted if it terminates with any return code.

Normally `cmlkeep` would restart the target process each time it ends, but under some circumstances the target process may need to be correctly shutdown or stopped with no desire to have it restarted. An example could be a network interface that is only required to be active during certain hours. At the end time it (and `cmlkeep`) can be gracefully shutdown by ending the program with a specific return code.

2.2.10 -r, –normal-restart-rc

`-r, --normal-restart-rc=<integer>`

This option specifies an integer and if the target process ends with a return code equal to this it will be silently restarted; that is the notification process is not executed. If this option is not specified then the notification process, if specified, will always be executed when the target processis restarted due to ending with any return code.

Normally if the target process ends unexpectedly then a notification is sent to indicate that some error occurred and the target process has been restarted. However a server may be restarted in order for it to read a new configuration and is programmed to terminate with a specific return code to indicate this. In this example `cmlkeep` would restart the target process without notification.

2.2.11 -E, –normal-exit-signal

`-E, --normal-exit-signal=<integer>`

This option specifies an integer and if the target process is terminated with a signal whose value is equal to this the restart cycle will be halted and `cmlkeep` will gracefully end. If this option is not specified then the target process will always be restarted if it is terminated through receiving a signal.

Normally `cmlkeep` would restart the target process each time it ends, but under some circumstances the target process may need to be correctly shutdown or stopped with no desire to have it restarted. An example could be a network interface that is only required to be active during certain hours. At the end time it (and `cmlkeep`) can be gracefully shutdown by ending the program with a specific signal.

2.2.12 -R, –normal-restart-signal

`-R, --normal-restart-signal=<integer>`

This option specifies an integer and if the target process is terminated with a signal whose value is equal to this it will be silently restarted; that is the notification process is not executed. If this option is not specified then the notification process, if specified, will always be executed when the target process is restarted due to a signal terminating it.

Normally if the target process ends unexpectedly then a notification is sent to indicate that some error occurred and the target process has been restarted. However a server may be restarted in order for it to read a new configuration and is programmed to terminate when it receives a specific signal. In this example `cmlkeep` would restart the target process without notification.

2.2.13 -H, –hup

`-H, --hup`

The shell invoking `cmlkeep` (and thus the target process) would normally send a `SIGHUP` signal to `cmlkeep` if it ends prematurely. As the default action of `SIGHUP` is to terminate the process, `cmlkeep` normally makes itself immune to hangups before invoking the target process. This means that there is no requirement to usually run `cmlkeep` with the `nohup` command.

If this behaviour is not required then specifying this option will leave the default behaviour of `SIGHUP` as is.

2.2.14 -k, –keep-log-spec

```
-k, --keep-log-spec={stderr|<am>(<obj>[,<opts>])}
```

Diagnostic and status messages generated by `cmlkeep` are, by default, written to standard error. Use this option to write these messages to a different location by using a `recio` access method open string. If the open of the (recio) destination fails the system automatically reverts to standard error.

2.2.15 -l, –log-file

```
-l, --log-file=<filename>|<pattern>
```

If this option is specified it should be a mask that when completed names a log file to which all the output (standard output and error) of the target process is written to. If it is not specified then all output from the target process is written using the Code Magus `rprintf`[1] whose default destination is standard error.

As a mask the default behaviour is to suffix the supplied pattern name with the process id and a serial number indicating the current segment of the log being processed. This behaviour can be changed by using the other options that pertain to the log file as shown below.

The log file is line buffered and each record (or line) is written out as a single entity.

2.2.16 -T, –log-name-strftime

```
-T, --log-name-strftime
```

This option allows a user to supply `--log-file` as a pattern that conforms to the C runtime functions `strftime()`. The caller can use `variables` to substitute date and times into the name. For a comprehensive list of substitution variables see the C manual on the target platform. A list of the more commonly ones follows.

- %d The day of the month as a decimal number (range 01 to 31).
- %H The hour as a decimal number using a 24-hour clock (range 00 to 23).
- %m The month as a decimal number (range 01 to 12).
- %M The minute as a decimal number (range 00 to 59).
- %s The number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC). (TZ)
- %S The second as a decimal number (range 00 to 60). (The range is up to 60 to allow for occasional leap seconds.)

- %Y The year as a decimal number including the century.
- %% A literal '%' character.

2.2.17 -B, –log-segment-bytes

`-B, --log-segment-bytes=<byte-count>`

If this option is specified then the log file is segmented or partitioned. This means that if the threshold of `byte-count` is exceeded while writing a record then the log file segment is closed and a new one is opened. One or more of the segmentation options can be specified together; each time a threshold is exceeded a new log segment is written.

2.2.18 -S, –log-segment-seconds

`-S, --log-segment-seconds=<seconds>`

If this option is specified then the log file is segmented or partitioned. This means that if the threshold of `seconds` since the segment was opened is exceeded while writing a record then the log file segment is closed and a new one is opened. One or more of the segmentation options can be specified together; each time a threshold is exceeded a new log segment is written.

2.2.19 -C, –log-segment-records

`-C, --log-segment-records=<records>`

If this option is specified then the log file is segmented or partitioned. This means that if the threshold of `records` is exceeded while writing a record then the log file segment is closed and a new one is opened. One or more of the segmentation options can be specified together; each time a threshold is exceeded a new log segment is written.

2.2.20 -I, –log-segment-signal

`-I, --log-segment-signal={0|<signum>}`

If this option is specified then the log file is segmented or partitioned. This means that if `cmlkeep` receives this signal it will, before the next record is written, close the current file segment and open a new one. One or more of the segmentation options can be specified together; each time a threshold is exceeded a new log segment is written.

2.2.21 -F, –log-flush-writes

```
-F, --log-flush-writes
```

If this option is specified then the each write to the RDW log file is flushed immediately. This effectively bypasses the standard stream buffering and is useful when running a job for diagnosis as the output is not lost in a buffer when the job aborts.

2.2.22 -O, –do-not-capture-output

```
-O, --do-not-capture-output
```

If this option is specified then no output from the target process will be captured. Note that any redirection in place during the invocation of `cmlkeep` to invoke the target process will affect the output of the target process.

2.2.23 -J, –print-job-directory

```
-J, --print-job-directory
```

If this option is set `cmlkeep` does not start a target process, even if other options are set, but will print to standard out the name of the directory that holds the job control information. This allows a system administrator to quickly find the directory used or for this option to be used in an automated script to find the directory.

An example of using this is in a machine start up script like `rc.local` on Linux. At start up it is best to clear all job information from this directory. This can only be done if the script can obtain the name used.

2.2.24 -v, –verbose

```
-v, --verbose
```

If this option is specified once verbose diagnostic output will be produced by `cmlkeep` and if specified twice then both verbose and trace diagnostic output will be produced.

2.2.25 -?, –help

```
-?, --help
```

If this option is specified then no processing takes place and a list of options and short descriptions are printed on the terminal.

2.2.26 –usage

`--usage`

As with `--help` but much more concise with no descriptions.

2.3 Starting a Job

A job is started by executing `cmlkeep` and passing it options for itself followed by '`--`' (dash dash), the target process fully qualified program name, followed in turn by the options to pass to the target process.

For example to start program `cmlserver` with the options '`--port 1024 --instances 10`' the following command could be used.

```
cmlkeep --job-name=CMLSERVER1 \
   --title="Code Magus Server Instance 1"
   --sleep=240 \
   --notify-command="mutt -s 'CMLINST 1 Error' help@codemagus.com" \
   --notify-file=/home/codemagus/email.txt \
   --restart-window=800 --restart-max=3 \
   --normal-exit-rc=0 --normal-restart-signal=10 \
   --normal-exit-signal=12 \
   --log-file=/home/codemagus/cmlserver1 \
   -- \
   cmlserver \
      --port 1024 --instances 10
```

The signal values 10 (`SIGUSR1`) and 12 (`SIGUSR2`) are convenient to use on Linux as a way of gracefully restarting or shutting down the target process as both signals have a default of terminating the process. Of course it depends on the application as to how it is shut down; others may trap different signals (like `SIGINT`) for shutdown or even ignore `SIGUSR1` and `SIGUSR2`, while others may have a networked interface to the application.

Also see the documentation for the `kill` program for the signal values on other Unix platforms.

If there is no need to change the default options to `cmlkeep` then the invocation could simply be

```
cmlkeep cmlserver --port 1024 --instances 10
```

In this instance it is not necessary to use the '`--`' to separate the options to `cmlkeep` and the name of the target process program. Here the job name will be `cmlserver`, it will have no title and no notification (other than `cmlkeep` writing to its log) will occur on a restart and no normal restart end return codes or signals will be defined.

2.4 Tuning excessive target process restarts

It is always difficult for software like `cmlkeep` to restart a target process that has failed or ended in a considered manner as `cmlkeep` may not have all the information required to make (an informed) decision.

There are two scenarios that are always in opposition as to the effect of the rules governing the restart of a target process. The first is that there is a fault in the target process and it repeatedly, and usually immediately, fails; and the second is that some outside influence causes the target process to fail at some point and then be unable to restart successfully. Both scenarios cause repeated restarts.

The first scenario requires that the restart process be stopped until the program is fixed and the second that the external problem be fixed while repeated restart of the target process is temporarily suspended or slowed down.

For example if a file system should fill up the impact on all jobs and the system could be catastrophic. In the first scenario, if a target process repeatedly fails on start up it may write one or more log messages and if this process continues it could fill the file system. Conversely if a rogue application fills the file system a target process, under the control of `cmlkeep`, may end when it receives an error while writing to an output file and then immediately fail on restarting; in this scenario it would be best to slow the restart rate down, but keep trying until the file system space problem is resolved.

The parameters `-m , --restart-max` and `-w, --restart-window` can be used to limit the number of successive immediate restarts a target process undergoes. However, this means that if `cmlkeep` attempts to restart the target process this often in the given time window it will give up and end. This is the same for both scenarios described above.

For the second scenario above it may not be desirable for `cmlkeep` to give up and end once the external problem is resolved it should still be trying to restart the target process.

To mitigate against `cmlkeep` giving up, the parameter `-y, --try-harder-to-restart` can be used. This is set to an integer; either negative, zero or positive. See the description of this parameter for the meaning when it is negative or zero. If it is positive then `cmlkeep` will allow this many successive windows of retries to be performed. It will also attempt to slow the process down by incrementing a wait between each successive window by adding the `--restart-window` number of seconds. It will count down the number of these successive windows and once it has used them all, will give up. If the target process is started up successfully and stays active for a long period then the countdown of these successive windows is reset to the original value.

Tuning these various parameters can be tricky and should be approached with care. It should be noted that along with these parameters the system administrator should also use the `--notify-command` to notify the relevant support personnel about any problems. Repeated immediate restarts and successive restart windows should be resolved

as quickly as possible.

3 Job Monitoring

3.1 Introduction

Code Magus `cmljobs` allows a user to monitor jobs started with `cmlkeep`.

3.2 Command line parameters

If `cmljobs` is started with the `--help` option it will only print help about the command line options with a short description of each as follows.

```
Usage: cmljobs [OPTION...]
  -j, --job-name=<string>     Check Status of a specific Job name
  -d, --delete                Remove the job reference (if job not running)
  -S, --silent                Do not write to stderr|stdout
  -v, --verbose               Verbose processing mode

Help options:
  -?, --help                  Show this help message
      --usage                 Display brief usage message
```

These parameters or options are further explained in the following sections.

3.2.1 -j, –job-name

`-j, --job-name=<string>`

This option is the name of the job as known to `cmljobs` and is the job name used when the job was started by `cmlkeep`. If this option is not given all jobs are checked.

3.2.2 -d, –delete

`-d, --delete`

This option can only be specified if `--job-name` is also specified. This option can be used to remove an orphaned reference to a target process if the target process is no longer running. This scenario could happen if both `cmlkeep` and the target process were terminated by some outside agent, leaving the reference to the job orphaned. `cmljobs` will attempt to remove the job reference after checking that the target process is not currently executing.

The job reference is a text file that holds the Process ID (PID) of the target process that was invoked by `cmlkeep`. There is a possibility that when a job reference is orphaned the system may reuse the same PID for another job not related to `cmlkeep`. In this instance `cmljobs` will not clean up the reference and manual intervention is required.

3.2.3 -S, –silent

`-S, --silent`

If this option is specified the `cmljobs` will produce little or no diagnostic output; If `--job-name` was not specified then only the list of known jobs is printed to standard out, but if `--job-name` was specified then no output is produced and only the return code will indicate the status of the job.

3.2.4 -v, –verbose

`-v, --verbose`

If this option is specified once verbose diagnostic output will be produced by `cmljobs` and if specified twice then both verbose and trace diagnostic output will be produced.

3.2.5 -?, –help

`-?, --help`

If this option is specified then no processing takes place and a list of options and short descriptions are printed on the terminal.

3.2.6 –usage

`--usage`

As with `--help` but much more concise with no descriptions.

3.3 Monitoring a Job

To get a list of jobs currently under the control of `cmlkeep` execute `cmljobs` and do not specify `--job-name`. This will then produce a list of jobs.

To get the status of a specific job then specify `--job-name` with the name of the job. The return code from `cmljobs` indicates the status of the job.

- Return Code 0. The job is known to to `cmlkeep` and currently running.
- Return code 4. The job is not known to `cmlkeep` and as such is not currently running.
- Return code 8. The job is known to `cmlkeep` but there is no target process running with the associated PID. In other words this is an orphaned job.

- Return code 16. A processing error has occurred. See the messages produced for an indication of why and what action can be taken.

4 Signalling a Job

4.1 Introduction

Code Magus `cmlkill` performs a similar function to the Linux / Unix `kill(1)` command in that it allows a user to send a signal. It is mostly used to send a signal to a job (the target process)that was started through `cmlkeep`, but may also be used to send a signal to `cmlkeep`. It differs to `kill` in that the user refers to the job name rather than the process ID of the target process, which helps to eliminate errors when sending critical signals to a job. All the signals that can be sent via the Unix command `kill` may be delivered via `cmlkill`.

Signals to `cmlkeep` are used mainly for causing a log switch for jobs running with logging and a log switch signal number specified.

4.2 Command line parameters

If `cmlkill` is started with the `--help` option it will only print help about the command line options with a short description of each as follows.

```
Usage: cmlkill [OPTION...]
  -j, --job-name=<string>             Job name (must be unique)
  -k, --cmlkeep                       Send the signal to the keep process (not
                                      the target process)
  -s, --signal={SIGKILL|<signal>}     Signal to send
  -S, --silent                        Do not write to stderr|stdout
  -v, --verbose                       Verbose processing mode

Help options:
  -?, --help                          Show this help message
      --usage                         Display brief usage message
```

These parameters or options are further explained in the following sections.

4.2.1 -j, –job-name

`-j, --job-name=<string>`

This mandatory option is the name of the job as known to `cmljobs` and is the job name used when the job was started by `cmlkeep`.

4.2.2 -k, –cmlkeep

`-k, --cmlkeep`

Send the signal to the keep process (not the target process). This option allows a signal to be delivered to the parent `cmlkeep` process and is most often used to cause a log switch.

4.2.3 -s, –signal

`-s, --signal={SIGKILL|<signal>}`

This mandatory option is the signal number or name that is to be sent to the target process of the job named in `--job-name`. The signal can be specified by number, short name (e.g. INT) or long name (e.g. `SIGINT`). See the Linux command '`kill' -l` for the names and numbers of the signals. These signals are specific to the operating system and may differ between Linux and Unix systems.

4.2.4 -S, –silent

`-S, --silent`

If this option is specified the `cmljobs` will produce no diagnostic output;

4.2.5 -v, –verbose

`-v, --verbose`

If this option is specified once verbose diagnostic output will be produced by `cmljobs` and if specified twice then both verbose and trace diagnostic output will be produced.

4.2.6 -?, –help

`-?, --help`

If this option is specified then no processing takes place and a list of options and short descriptions are printed on the terminal.

4.2.7 –usage

`--usage`

As with `--help` but much more concise with no descriptions.

4.3 Signalling Examples

To send an interrupt signal to the target process of a job called `BATCHRUN` use the following command:

```
cmlkill -j BATCHRUN -s INT
..or..
cmlkill -j BATCHRUN -s sigint
..or..
cmlkill -j BATCHRUN -s 2
```

To cause a log switch of the `cmlkeep` log where the signal is specified as 10 (or `SIGUSR1`) use the following command:

```
cmlkill -k -j BATCHRUN -s USR1
```

References

[1] rprintf API: User Guide and Reference Version 1. CML Document CML00002-01, Code Magus Limited, July 2008. PDF.

[2] binary: Fixed and Variable Length Record Stream Access Method Version 1. CML Document CML00005-01, Code Magus Limited, July 2008. PDF.