Step 3: Rotate the log file. Rotate the log file by issuing the `logRotate` command from the `admin` database in a `mongo` shell:

```
use admin
db.runCommand( { logRotate : 1 } )
```

You should rename the log file using an external process, following the typical Linux/Unix log rotate behavior.

Syslog Log Rotation

New in version 2.2.

With syslog log rotation, `mongod` sends log data to the syslog rather than writing it to a file.

Step 1: Start a `mongod` instance with the *`--syslog`* option

```
mongod --syslog
```

Do not include *`--logpath`*. Since *`--syslog`* tells `mongod` to send log data to the syslog, specifying a *`--logpath`* will causes an error.

To specify the facility level used when logging messages to the syslog, use the *`--syslogFacility`* option or `systemLog.syslogFacility` configuration setting.

Step 2: Rotate the log. Store and rotate the log output using your systems default log rotation mechanism.

Forcing a Log Rotation with `SIGUSR1`

For Linux and Unix-based systems, you can use the `SIGUSR1` signal to rotate the logs for a single process, as in the following:

```
kill -SIGUSR1 <mongod process id>
```

Manage Journaling

MongoDB uses *write ahead logging* to an on-disk *journal* to guarantee *write operation* (page 73) durability and to provide crash resiliency. Before applying a change to the data files, MongoDB writes the change operation to the journal. If MongoDB should terminate or encounter an error before it can write the changes from the journal to the data files, MongoDB can re-apply the write operation and maintain a consistent state.

Without a journal, if `mongod` exits unexpectedly, you must assume your data is in an inconsistent state, and you must run either *repair* (page 263) or, preferably, *resync* (page 633) from a clean member of the replica set.

With journaling enabled, if `mongod` stops unexpectedly, the program can recover everything written to the journal, and the data remains in a consistent state. By default, the greatest extent of lost writes, i.e., those not made to the journal, are those made in the last 100 milliseconds. See `commitIntervalMs` for more information on the default.

With journaling, if you want a data set to reside entirely in RAM, you need enough RAM to hold the data set plus the "write working set." The "write working set" is the amount of unique data you expect to see written between re-mappings of the private view. For information on views, see *Storage Views used in Journaling* (page 309).

Important: Changed in version 2.0: For 64-bit builds of `mongod`, journaling is enabled by default. For other platforms, see `storage.journal.enabled`.