

Logging Configuration



- [The \[log\] section](#)
- [Quick samples](#)
 - [Default \(empty\) Configuration](#)
 - [Disable Router logs and record some access logs to file](#)
 - [Set different log levels for different modes](#)
- [Log outputs \(mode and writer\)](#)
 - [Common configuration](#)
 - [EXPRESSION](#)
 - [EXCLUSION](#)
 - [FLAGS](#)
 - [Console mode](#)
 - [journald mode](#)
 - [File mode](#)
 - [Conn mode](#)
 - [The "Router" logger](#)
 - [The "XORM" logger](#)
 - [The "Access" logger](#)
 - [The ACCESS_LOG_TEMPLATE](#)
- [Releasing-and-Reopening, Pausing and Resuming logging](#)
- [Adding and removing logging whilst Forgejo is running](#)
- [Using logrotate instead of built-in log rotation](#)

The logging configuration of Forgejo mainly consists of 3 types of components:

- The `[log]` section for general configuration.
- `[log.<mode-name>]` sections for the configuration of different log writers to output logs, aka: "writer mode", the mode name is also used as "writer name".
- The `[log]` section can also contain sub-logger configurations following the key schema `logger.<logger-name>.<CONFIG-KEY>`.

There is a fully functional log output by default, so it is not necessary to define one.

The `[log]` section

Configuration of logging facilities in Forgejo happens in the `[log]` section and its subsections.

The top level section `[log]` has the following properties:

- `ROOT_PATH`: (Default: [AppWorkPath/log](#)): Base path for log files.
- `MODE`: (Default: **console**): List of log outputs to use for the Default logger.
- `LEVEL`: (Default: **Info**): Least severe log events to persist, case-insensitive. Possible values, sorted by increasing severity, are: `Trace`, `Debug`, `Info`, `Warn`, `Error`, `Fatal`.
- `STACKTRACE_LEVEL`: (Default: **None**): Least severe events to print stacktrace for. Possible values are the same as for the `LEVEL` option.

The `[log]` section can contain sub-logger modes as well:

- `LOGGER_DEFAULT_MODE`: (Default: `,`): List of log outputs to use for the default logger.
- `LOGGER_ROUTER_MODE`: (Default: `,`): List of log outputs to use for the Router logger.
- `LOGGER_ACCESS_MODE`: (Default: **<empty>**): List of log outputs to use for the Access logger. By default, the access logger is disabled.
- `LOGGER_XORM_MODE`: (Default: `,`): List of log outputs to use for the XORM logger.

Setting a sub-logger's mode to a comma (`,`) makes it use the global `MODE`.

Quick samples

Default (empty) Configuration

The empty configuration is equivalent to default:

```
[log]
ROOT_PATH = %(GITEA_WORK_DIR)/log
MODE = console
LEVEL = Info
STACKTRACE_LEVEL = None
LOGGER_DEFAULT_MODE = ,
LOGGER_ROUTER_MODE = ,
LOGGER_XORM_MODE = ,
```

```
LOGGER_ACCESS_MODE =  
  
; this is the config options of "console" mode (used by MODE=console above)  
[log.console]  
MODE = console  
FLAGS = stdflags  
PREFIX =  
COLORIZE = true
```

This is equivalent to sending all logs to the console, with default Golang log being sent to the console log too.

This is only a sample, and it is the default, do not need to write it into your configuration file.

Disable Router logs and record some access logs to file

The Router logger is disabled, the access logs ($\geq$ Warn) goes into `access.log`:

```
[log]  
LOGGER_ROUTER_MODE =  
LOGGER_ACCESS_MODE = access-file  
  
[log.access-file]  
MODE = file  
LEVEL = Warn  
FILE_NAME = access.log
```

Set different log levels for different modes

Default logs ($\geq$ Warn) goes into `gitea.log`, while Error logs goes into `file-error.log`:

```
[log]  
LEVEL = Warn  
MODE = file, file-error  
  
; by default, the "file" mode will record logs to %(log.ROOT_PATH)/gitea.log, so we don't need  
to set it
```

```
; [log.file]
; by default, the MODE (actually it's the output writer of this logger) is taken from the
section name, so we don't need to set it either
; MODE = file

[log.file-error]
MODE = file
LEVEL = Error
FILE_NAME = file-error.log
```

Log outputs (mode and writer)

Forgejo provides the following log output writers:

- `console` - Log to `stdout` (or `stderr` if it is set in the config).
- `file` - Log to a file.
- `conn` - Log to a socket (network or unix).

Common configuration

Certain configuration is common to all modes of log output:

- `MODE` is the mode of the log output writer. It will default to the mode name in the ini section. Thus `[log.console]` will default to `MODE = console`.
- `LEVEL` is the lowest level that this output will log.
- `STACKTRACE_LEVEL` is the lowest level that this output will print a stacktrace.
- `COLORIZE` will default to `true` for `console` if writing to the terminal (see below), otherwise it will default to `false`.

EXPRESSION

`EXPRESSION` represents a regular expression that log events must match to be logged by the output writer. Either the log message, (with colors removed), must match or the `longfilename:linenumber:functionname` must match. NB: the whole message or string doesn't need to completely match.

Please note this expression will be run in the writer's goroutine but not the logging event goroutine.

EXCLUSION

`EXCLUSION` represents a regular expression that log events must not match in order to be logged by the output writer. Either the log message (with colors removed) or the

`longfilename:linenumber:functionname` must not match the exclusion pattern. NB: the whole message or string doesn't need to completely match.

Please note this exclusion will be run in the writer's goroutine but not the logging event goroutine.

For example, you can collect logs from the Router, but requests to `/api/healthz` and `/api/metrics` are excluded:

```
[log.router]
MODE   = console
LEVEL  = INFO
EXCLUSION = (\api\healthz|\metrics)
```

FLAGS

`FLAGS` represents the preceding logging context information that is printed before each message. It is a comma-separated string set. The order of values does not matter.

It defaults to `stdflags` (= `date,time,medfile,shortfuncname,levelinitial`). Except for the access log which defaults to `none`

Possible values are:

- `none` or `,` - No flags.
- `date` - the date in the local time zone: `2009/01/23`.
- `time` - the time in the local time zone: `01:23:23`.
- `microseconds` - microsecond resolution: `01:23:23.123123`. Assumes time.
- `longfile` - full file name and line number: `/a/b/c/d.go:23`.
- `shortfile` - final file name element and line number: `d.go:23`.
- `funcname` - function name of the caller: `runtime.Caller()`.
- `shortfuncname` - last part of the function name. Overrides `funcname`.
- `utc` - if date or time is set, use UTC rather than the local time zone.
- `levelinitial` - initial character of the provided level in brackets eg. `[I]` for info.
- `level` - level in brackets `[INFO]`.
- `levelprefix` - journald-style log level prefix, such as `<5>` for "info".
- `gopid` - the Goroutine-PID of the context.
- `medfile` - last 20 characters of the filename - equivalent to `shortfile,longfile`.
- `stdflags` - equivalent to `date,time,medfile,shortfuncname,levelinitial`.
- `journaldflags` - equivalent to `levelprefix`.

Console mode

In this mode the logger will forward log messages to the stdout and stderr streams attached to the Forgejo process.

For loggers in console mode, `COLORIZE` will default to `true` when appropriate:

- not on Windows, if the output is connected to a tty;
- on Windows, if the output is connected to either a Windows terminal that can be switched into ANSI mode (Windows 10+), or a Cygwin/MSYS terminal.

Settings:

- `STDERR`: **false**: Whether the logger should print to `stderr` instead of `stdout`.

journald mode

As an extension to the console mode, if Forgejo detects that stdout and/or stderr are connected to systemd-journald (which will happen automatically when Forgejo is running under systemd), the defaults will change:

- `STDERR` will default to **true** if stderr is connected to systemd-journald;
- `FLAGS` will default to `journaldflags` if the chosen stream is connected to systemd-journald.

When `FLAGS = journaldflags` is used, instead of decorating log entries with color or a textual representation of severity (e.g. `[INFO]` or `[ERROR]`), Forgejo will annotate each log entry with a machine-readable log level prefix that is captured by journald, enabling native filtering capabilities (e.g. run `journalctl -u forgejo -p err` to only show errors or worse).

In short, if Forgejo is running under systemd, it is recommended to leave the logging configuration at their defaults, which will use `MODE = console` and adjust formatting to reduce clutter and enable native log level filtering.

File mode

In this mode the logger will save log messages to a file.

Settings:

- `FILE_NAME`: The file to write the log events to, relative to `ROOT_PATH`, Default to `%(ROOT_PATH)/gitea.log`. Exception: access log will default to `%(ROOT_PATH)/access.log`.
- `MAX_SIZE_SHIFT`: **28**: Maximum size shift of a single file. 28 represents 256Mb. For details see below.
- `LOG_ROTATE` **true**: Whether to rotate the log files. If `false`, log files will always be appended.
- `DAILY_ROTATE`: **true**: Whether to rotate logs daily.
- `MAX_DAYS`: **7**: Delete rotated log files after this number of days.
- `COMPRESS`: **true**: Whether to compress old log files by default with gzip.

- `COMPRESSION_LEVEL`: **-1**: Compression level. For details see below.

`MAX_SIZE_SHIFT` defines the maximum size of a file by left shifting 1 the given number of times (`1 << x`).

The useful values of `COMPRESSION_LEVEL` are from 1 to (and including) 9, where higher numbers mean better compression. Beware that better compression might come with higher resource usage. Must be preceded with a `-` sign.

Conn mode

In this mode the logger will send log messages over a network socket.

Settings:

- `ADDR`: **:7020**: Sets the address to connect to.
- `PROTOCOL`: **tcp**: Set the protocol, either "tcp", "unix" or "udp".
- `RECONNECT`: **false**: Try to reconnect when connection is lost.
- `RECONNECT_ON_MSG`: **false**: Reconnect host for every single message.

The "Router" logger

The Router logger logs the following message types when Forgejo's route handlers work:

- `started` messages will be logged at TRACE level.
- `polling`/`completed` routers will be logged at INFO. Exception: "/assets" static resource requests are also logged at TRACE.
- `slow` routers will be logged at WARN.
- `failed` routers will be logged at WARN.

The "XORM" logger

To make XORM outputs SQL logs, the `LOG_SQL` in `[database]` section should also be set to `true`.

The "Access" logger

The Access logger provides a NCSA Common Log compliant log format. It's highly configurable but caution should be taken when changing its template. The main benefit of this logger is that Forgejo can now log accesses in a standard log format so standard tools may be used.

You can enable this logger using `LOGGER_ACCESS_MODE = ...`.

If desired the format of the Access logger can be changed by changing the value of the `ACCESS_LOG_TEMPLATE`.

Please note, the access logger will log at `INFO` level, setting the `LEVEL` of this logger to `WARN` or above will result in no access logs.

The `ACCESS_LOG_TEMPLATE`

This value represents a go template. Its default value is

```
{{.Ctx.RemoteHost}} - {{.Identity}} {{.Start.Format "[02/Jan/2006:15:04:05 -0700]" }}  
"{{.Ctx.Req.Method}} {{.Ctx.Req.URL.RequestURI}} {{.Ctx.Req.Proto}}"  
{{.ResponseWriter.Status}} {{.ResponseWriter.Size}} "{{.Ctx.Req.Referer}}"  
"{{.Ctx.Req.UserAgent}}"
```

The template is passed following options:

- `Ctx` is the `context.Context`.
- `Identity` is the `SignedUserName` or `"-"` if the user is not logged in.
- `Start` is the start time of the request.
- `ResponseWriter` is the `http.ResponseWriter`.

Caution must be taken when changing this template as it runs outside of the standard panic recovery trap. The template should also be as simple as it runs for every request..

Releasing-and-Reopening, Pausing and Resuming logging

If you are running on Unix you may wish to release-and-reopen logs in order to use `logrotate` or other tools. It is possible force Forgejo to release and reopen its logging files and connections by sending `SIGUSR1` to the running process, or running `forgejo manager logging release-and-reopen`.

Alternatively, you may wish to pause and resume logging - this can be accomplished through the use of the `forgejo manager logging pause` and `forgejo manager logging resume` commands. Please note that whilst logging is paused log events below `INFO` level will not be stored and only a limited number of events will be stored. Logging may block, albeit temporarily, slowing Forgejo considerably whilst paused - therefore it is recommended that pausing only done for a very short period of time.

Adding and removing logging whilst Forgejo is running

It is possible to add and remove logging whilst Forgejo is running using the `forgejo manager logging add` and `remove` subcommands. This functionality can only adjust running log systems and cannot be used to start the access or router loggers if they were not already initialized. If you wish to start these systems you are advised to adjust the `app.ini` and (gracefully) restart the Forgejo service.

The main intention of these commands is to easily add a temporary logger to investigate problems on running systems where a restart may cause the issue to disappear.

Using `logrotate` instead of built-in log rotation

Forgejo includes built-in log rotation, which should be enough for most deployments. However, if you instead want to use the `logrotate` utility:

- Disable built-in log rotation by setting `LOG_ROTATE` to `false` in your `app.ini`.
- Install `logrotate`.
- Configure `logrotate` to match your deployment requirements, see `man 8 logrotate` for configuration syntax details. In the `postrotate/endscript` block send Forgejo a `USR1` signal via `kill -USR1` or `kill -10` to the `forgejo` process itself, or run `forgejo manager logging release-and-reopen` (with the appropriate environment). Ensure that your configurations apply to all files emitted by Forgejo loggers as described in the above sections.
- Always do `logrotate /etc/logrotate.conf --debug` to test your configurations.
- If you are using docker and are running from outside the container you can use `docker exec -u $OS_USER $CONTAINER_NAME sh -c 'forgejo manager logging release-and-reopen'` or `docker exec $CONTAINER_NAME sh -c '/bin/s6-svc -1 /etc/s6/gitea/'` or send `USR1` directly to the Forgejo process itself.

The next `logrotate` jobs will include your configurations, so no restart is needed. You can also immediately reload `logrotate` with `logrotate /etc/logrotate.conf --force`.

☐☐ #3

☐ cocobook-sync ☐☐ 2026-10-28 16:41:24 UTC

☐ cocobook-sync ☐☐ 2026-10-29 01:09:33 UTC