# Contributing To Kamailio #

\*First, thank you for taking the time to contribute to Kamailio project!\*

The following is a set of guidelines for contributing to Kamailio sources and

documentation. Kamailio source tree is hosted in the [Kamailio Organization](https://github.com/kamailio) on GitHub.

These are intended to be more like guidelines to keep everything consistent and

coherent, not very strict rules. Use your best judgment and feel free to propose

changes to this document in a pull request.

### Table Of Contents ###

\* [Overview](#overview)

\* [Contributing Code Or Content](#contributing-code-or-content)

\* [Basic Rules](#basic-rules)

\* [Commit Message Rules](#commit-message-rules)

\* [Commit Message Format](#commit-message-format)

\* [Commit Message Content](#commit-message-content)

\* [Commit Message Examples](#commit-message-examples)

\* [See Also](#see-also)

\* [Reporting Issues](#reporting-issues)

\* [License](#license)

\* [License Of New Code Contributions](#license-of-new-code-contributions)

\* [Further Assistance](#further-assistance)

## Overview ##

Kamailio is a community managed project, with developers world wide. Any

contribution to code or documentation is very welcome and appreciated.

In order to be easily able to track the changes and have a coherent ChangLog

and commit history, there are several \*rules\* required for each contribution.

## Contributing Code Or Content ##

### Basic Rules ###

\* github pull request is the favorited mechanism to submit contributions

(patches)

\* make a pull request against \*\*master branch\*\*

\* commit can be later backported to stable branch(es)

\* make a pull request for each new feature

\* e.g., if you add a feature to usrloc module and an unrelated feature

to auth module, then make two pull requests

\* it is ok (and sometime recommended) to have more than one commit per pull request

\* make a commit for each affected component. A component is considered to be:

\* the core

\* an internal library (code inside subfolder lib/)

\* a module (code inside subfolder modules/)

\* a tool (code inside subfolder utils/)

\* an example or main configs (files inside subfolders etc/ or misc/examples/)

\* commit message format \*\*has to follow the rules\*\* specified in the next section

\* commit message content \*\*has to follow the rules\*\* specified in the next section

\* changes to the \*\*README\*\* file of a module \*\*must not\*\* be done directly in that

file. Instead, edit the xml files located in \*\*modules/modname/doc/\*\* folder

\* to regenerate the README, run \*\*make modules-readme modules=modules/modname\*\*

\* docbook utils and xsl packages are needed for the above command to work

\* it is only necessary to modify and commit the xml doc file, the \*\*README\*\*

will be regenerated by a automatic script that is executed every few hours

\* so if you modify an existing module \*\*README\*\* don't commit the changes

of this file to the git repository

\* if you create a new module that includes also a \*\*README\*\* file, you

need to commit the README to the git repository one time

\* code \*\*should\*\* be formatted with \*\*clang-format\*\* or to match the style of

the component that the commit applies to. The `.clang-format` file is part of

Kamailio source code tree, in the root folder.

\* new function exports to the configuration \*\*must\*\* be done both as native

configuration export and KEMI function export.

\* new introduced module parameters, functions and RPC commands \*\*must\*\*

be documented in the module documentation.

\* new introduced pseudo-variables and transformations \*\*must\*\* be

documented in the wiki.

### Commit Message Rules ###

#### Commit Message Format ####

Please create the commit messages following the GIT convention:

\* start with one short line, preferably less then 50 chars summarizing the

changes (this is referred later as "first line of the commit message")

\* then one empty line

\* then a more detailed description (the "commit message content")

#### First Line Of The Commit Message ####

Think of the first line as of an email "Subject" line. In fact it will be used

as "Subject" in the generated commit emails and it will also be used when

generating the Changelog (e.g. git log --pretty=oneline).

The first line (subject has to contain meaningful text about what that commit

does, do not put just a reference to bug tracker or pull request items

Please start always with the prefix of the component (subsystem) that is modified

by the commit, for example:

\* `core`: more fixup helper functions

\* `core`: tcp - support for haproxy protocol

\* `core`: mem - added faster malloc

\* `modname`: support for foo rfc extension

\* `usrloc`: support for gruu rfc extension

\* `lib`: srutils - critical bug fix for abc case

\* `kamctl`: added support for management of module xyz

At the end of the first line some CI flags can be added. Available at this

moment:

\* `[skip ci]` - skip continous integration builds for source code, recommended

to be added when updating documentation, example configs or other utilities.

Example:

\* `msilo: docs - updated example for m\_dump() function [skip ci]`

#### Commit Message Content ####

\* commit message must describe the changes done by the patch

\* other details (e.g., how to reproduce, backtrace, sip packets, ...) belong

to content (comments) of the pull request. Example of a full commit message:

```

core: added latency\_limit\_cfg global parameter

- print execution time for configuration script only if it exceeds this value

- default is 0 - print always (behaviour so far)

- it is printed to latency\_cfg\_log level

```

\* avoid emoticons and non-technical statements in commit messages

\* e.g., if it was a feature request by John Smith, don't mention that in

commit message, especially don't write it owns you now a beer

\* credits can be given within commit message as a short statement, mentioning

the name of the person or entity

\* for commits introducing a new module, credits must not be included in the

commit message, being expected that the respective entity will own the

copyright and it is reflected in the README or copyright header of each file

\* when the case, make references in the commit body (not in the subject/first line)

to the items on bug tracker or pull requests, using GH #XYZ

-- replace XYZ with issue number id. Example:

```

dialplan: basic safety for concurrent rpc reload

- reported by GH #1874

```

\* commits related to reports by static analyzers or other tools must describe

what was fixed or changed. The tool, if a well known one, can be mentioned in

the body of the commit message, after the technical details presenting the

changes. For example, do not use commit messages like:

```

...: fix for whatever-tool reports

```

\* do not reference non-public resources (e.g., private links, id of non-public

static analyzer reports, ...). For example, do not use commit messages like:

```

...: fix for whatever-tool report #1234

```

```

...: fix for http://private-tracker.lab/1234

```

\* do not mention any developer name or yourself when fixing an issue introduced

by an old commit done by that developer or you. That commit can be referenced

by hash id.

#### Commit Message Examples ####

\* change to usrloc module from modules

```

usrloc: fixed name conflict

- destroy\_avps() renamed to reg\_destroy\_avps() to avoid conflicts

with the usr\_avp.h version

```

\* change to core

```

core: loadpath can now use a list of directories

- loadpath can use a list of directories separated by ':',

e.g.: loadpath "modules:modules\_s:modules\_k".

First match wins (e.g. for loadmodule "textops" if

modules/textops.so or modules/textops/textops.so exists, it will

be loaded and the search will stop).

```

#### See Also ####

\* [Creating Good Commit Messages](http://www.kernel.org/pub/software/scm/git/docs/user-manual.html#creating-good-commit-messages)

\* http://www.tpope.net/node/106

The above content about commit message format is taken from Kamailio wiki page:

\* https://www.kamailio.org/wiki/devel/git-commit-guidelines

\* it is recommended you read that one as well.

### Developer Access ###

\* developer access (commit rights) to Kamailio GIT repository is granted to

people that contribute relevant components (e.g., modules) or have consistent

contributions over a long interval of time

\* each developer has to create an account on github.com portal. The

`developerid` is the username on github.com portal

\* after getting developer access, it is still recommended to use pull request

for commits done to other components of Kamailio, to allow the main developer

of the component as well as the other developers to review the changes

\* commits to own components can be pushed directly, without a pull request.

However, if the developer wants other people to review the changes, using a

pull request is the way to do it

\* personal branches of developers done inside Kamailio GIT repository must be

prefixed with `developerid/`, e.g., `alice/new-feature`. Do not use just

`new-feature` or `alice-new-feature` or other variant without `developerid/`

\* the [Contributions Basic Rules](#basic-rules) from the sections above have

to be followed as well after getting developer access

## Reporting Issues ##

Whenever reporting an issue, along with the description of the problems, try to

include following details:

\* kamailio version you are using

\* the output of: \*\*kamailio -v\*\*

\* the operating system being used

\* the CPU architecture

Always useful to have:

\* whenever there is a crash with a corefile, send the backtrace

\* the output of \*\*bt full\*\* in \*\*gbd\*\*

\* log messages printed by kamailio in syslog file

\* \*pcap\* or \*ngrep\* capture of SIP packets causing the issue

\* config file snippets which expose the issues

Note: replace any sensitive information in the content you add to the issue

(e.g., passwords in modparams can be replaced with xyz, each IP address can be

replaced with tokens like a.b.c.d, f.g.h.j).

## License ##

Kamailio Main License: \*GPLv2\*.

Each source code file refers to the license and copyright details in the top

of the file. Most of the code is licensed under GPLv2 (with the "any later

version" clause), some parts of the code are licensed under BSD.

### License Of New Code Contributions ###

New contributions to the core and several main modules (auth, corex, sl, tls,

tm) have to be done under the BSD license. New contributions under the GPL must

grant the GPL-OpenSSL linking exception. Contributions to existing components

released under BSD must be done under BSD as well.

## Further Assistance ###

For any question, do not hesitate to contact other developers via mailing list:

\* \*\*[sr-dev [at] lists.kamailio.org](http://lists.kamailio.org/cgi-bin/mailman/listinfo/sr-dev)\*\*