# <a id="contributing"></a> Contributing

Icinga is an open source project and lives from your ideas and contributions.

There are many ways to contribute, from improving the documentation, submitting

bug reports and features requests or writing code to add enhancements or fix bugs.

#### Table of Contents

1. [Introduction](#contributing-intro)

2. [Fork the Project](#contributing-fork)

3. [Branches](#contributing-branches)

4. [Commits](#contributing-commits)

5. [Pull Requests](#contributing-pull-requests)

6. [Testing](#contributing-testing)

7. [Source Code Patches](#contributing-patches-source-code)

8. [Documentation Patches](#contributing-patches-documentation)

9. [Contribute CheckCommand Definitions](#contributing-patches-itl-checkcommands)

10. [Review](#contributing-review)

## <a id="contributing-intro"></a> Introduction

Please consider our [roadmap](https://github.com/Icinga/icinga2/milestones) and

[open issues](https://github.com/icinga/icinga2/issues) when you start contributing

to the project.

Issues labeled with [help wanted](https://github.com/Icinga/icinga2/labels/help%20wanted) or

[good first issue](https://github.com/Icinga/icinga2/labels/good%20first%20issue) will

help you get started more easily.

Before starting your work on Icinga 2, you should [fork the project](https://help.github.com/articles/fork-a-repo/)

to your GitHub account. This allows you to freely experiment with your changes.

When your changes are complete, submit a [pull request](https://help.github.com/articles/using-pull-requests/).

All pull requests will be reviewed and merged if they suit some general guidelines:

\* Changes are located in a topic branch

\* For new functionality, proper tests are written

\* Changes should follow the existing coding style and standards

Please continue reading in the following sections for a step by step guide.

## <a id="contributing-fork"></a> Fork the Project

[Fork the project](https://help.github.com/articles/fork-a-repo/) to your GitHub account

and clone the repository:

```

git clone git@github.com:dnsmichi/icinga2.git

cd icinga2

```

Add a new remote `upstream` with this repository as value.

```

git remote add upstream https://github.com/icinga/icinga2.git

```

You can pull updates to your fork's master branch:

```

git fetch --all

git pull upstream HEAD

```

Please continue to learn about [branches](CONTRIBUTING.md#contributing-branches).

## <a id="contributing-branches"></a> Branches

Choosing a proper name for a branch helps us identify its purpose and possibly

find an associated bug or feature.

Generally a branch name should include a topic such as `bugfix` or `feature` followed

by a description and an issue number if applicable. Branches should have only changes

relevant to a specific issue.

```

git checkout -b bugfix/service-template-typo-1234

git checkout -b feature/config-handling-1235

```

Continue to apply your changes and test them. More details on specific changes:

\* [Source Code Patches](#contributing-patches-source-code)

\* [Documentation Patches](#contributing-patches-documentation)

\* [Contribute CheckCommand Definitions](#contributing-patches-itl-checkcommands)

## <a id="contributing-commits"></a> Commits

Once you've finished your work in a branch, please ensure to commit

your changes. A good commit message includes a short topic, additional body

and a reference to the issue you wish to solve (if existing).

Fixes:

```

Fix problem with notifications in HA cluster

There was a race condition when restarting.

refs #4567

```

Features:

```

Add ITL CheckCommand printer

Requires the check\_printer plugin.

refs #1234

```

You can add multiple commits during your journey to finish your patch.

Don't worry, you can squash those changes into a single commit later on.

## <a id="contributing-pull-requests"></a> Pull Requests

Once you've commited your changes, please update your local master

branch and rebase your bugfix/feature branch against it before submitting a PR.

```

git checkout master

git pull upstream HEAD

git checkout bugfix/notifications

git rebase master

```

Once you've resolved any conflicts, push the branch to your remote repository.

It might be necessary to force push after rebasing - use with care!

New branch:

```

git push --set-upstream origin bugfix/notifications

```

Existing branch:

```

git push -f origin bugfix/notifications

```

You can now either use the [hub](https://hub.github.com) CLI tool to create a PR, or nagivate

to your GitHub repository and create a PR there.

The pull request should again contain a telling subject and a reference

with `fixes` to an existing issue id if any. That allows developers

to automatically resolve the issues once your PR gets merged.

```

hub pull-request

<a telling subject>

fixes #1234

```

Thanks a lot for your contribution!

### <a id="contributing-rebase"></a> Rebase a Branch

If you accidentally sent in a PR which was not rebased against the upstream master,

developers might ask you to rebase your PR.

First off, fetch and pull `upstream` master.

```

git checkout master

git fetch --all

git pull upstream HEAD

```

Then change to your working branch and start rebasing it against master:

```

git checkout bugfix/notifications

git rebase master

```

If you are running into a conflict, rebase will stop and ask you to fix the problems.

```

git status

both modified: path/to/conflict.cpp

```

Edit the file and search for `>>>`. Fix, build, test and save as needed.

Add the modified file(s) and continue rebasing.

```

git add path/to/conflict.cpp

git rebase --continue

```

Once succeeded ensure to push your changed history remotely.

```

git push -f origin bugfix/notifications

```

If you fear to break things, do the rebase in a backup branch first and later replace your current branch.

```

git checkout bugfix/notifications

git checkout -b bugfix/notifications-rebase

git rebase master

git branch -D bugfix/notifications

git checkout -b bugfix/notifications

git push -f origin bugfix/notifications

```

### <a id="contributing-squash"></a> Squash Commits

> \*\*Note:\*\*

>

> Be careful with squashing. This might lead to non-recoverable mistakes.

>

> This is for advanced Git users.

Say you want to squash the last 3 commits in your branch into a single one.

Start an interactive (`-i`) rebase from current HEAD minus three commits (`HEAD~3`).

```

git rebase -i HEAD~3

```

Git opens your preferred editor. `pick` the commit in the first line, change `pick` to `squash` on the other lines.

```

pick e4bf04e47 Fix notifications

squash d7b939d99 Tests

squash b37fd5377 Doc updates

```

Save and let rebase to its job. Then force push the changes to the remote origin.

```

git push -f origin bugfix/notifications

```

## <a id="contributing-testing"></a> Testing

Please follow the [documentation](https://icinga.com/docs/icinga2/snapshot/doc/21-development/#test-icinga-2)

for build and test instructions.

You can help test-drive the latest Icinga 2 snapshot packages inside the

[Icinga 2 Vagrant boxes](https://github.com/icinga/icinga-vagrant).

## <a id="contributing-patches-source-code"></a> Source Code Patches

Icinga 2 can be built on Linux/Unix nodes and Windows clients. In order to develop patches for Icinga 2,

you should prepare your own local build environment and know how to work with C++.

Please follow the [development documentation](https://icinga.com/docs/icinga2/latest/doc/21-development/)

for development environments, the style guide and more advanced insights.

## <a id="contributing-patches-documentation"></a> Documentation Patches

The documentation is written in GitHub flavored [Markdown](https://guides.github.com/features/mastering-markdown/).

It is located in the `doc/` directory and can be edited with your preferred editor. You can also

edit it online on GitHub.

```

vim doc/2-getting-started.md

```

In order to review and test changes, you can install the [mkdocs](http://www.mkdocs.org) Python library.

```

pip install mkdocs

```

This allows you to start a local mkdocs viewer instance on http://localhost:8000

```

mkdocs serve

```

Changes on the chapter layout can be done inside the `mkdocs.yml` file in the main tree.

There also is a script to ensure that relative URLs to other sections are updated. This script

also checks for broken URLs.

```

./doc/update-links.py doc/\*.md

```

## <a id="contributing-patches-itl-checkcommands"></a> Contribute CheckCommand Definitions

The Icinga Template Library (ITL) and its plugin check commands provide a variety of CheckCommand

object definitions which can be included on-demand.

Advantages of sending them upstream:

\* Everyone can use and update/fix them.

\* One single place for configuration and documentation.

\* Developers may suggest updates and help with best practices.

\* You don't need to care about copying the command definitions to your satellites and clients.

#### <a id="contributing-itl-checkcommands-start"></a> Where do I start?

Get to know the check plugin and its options. Read the general documentation on how to integrate

your check plugins and how to create a good CheckCommand definition.

A good command definition uses:

\* Command arguments including `value`, `description`, optional: `set\_if`, `required`, etc.

\* Comments `/\* ... \*/` to describe difficult parts.

\* Command name as prefix for the custom attributes referenced (e.g. `disk\_`)

\* Default values

\* If `host.address` is involved, set a custom attribute (e.g. `ping\_address`) to the default `$address$`. This allows users to override the host's address later on by setting the custom attribute inside the service apply definitions.

\* If the plugin is also capable to use ipv6, import the `ipv4-or-ipv6` template and use `$check\_address$` instead of `$address$`. This allows to fall back to ipv6 if only this address is set.

\* If `set\_if` is involved, ensure to specify a sane default value if required.

\* Templates if there are multiple plugins with the same basic behaviour (e.g. ping4 and ping6).

\* Your love and enthusiasm in making it the perfect CheckCommand.

#### <a id="contributing-itl-checkcommands-overview"></a> I have created a CheckCommand, what now?

Icinga 2 developers love documentation. This isn't just because we want to annoy anyone sending a patch,

it's a matter of making your contribution visible to the community.

Your patch should consist of 2 parts:

\* The CheckCommand definition.

\* The documentation bits.

[Fork the repository](https://help.github.com/articles/fork-a-repo/) and ensure that the master branch is up-to-date.

Create a new fix or feature branch and start your work.

```

git checkout -b feature/itl-check-printer

```

#### <a id="contributing-itl-checkcommands-add"></a> Add CheckCommand Definition to Contrib Plugins

There already exists a defined structure for contributed plugins. Navigate to `itl/plugins-contrib.d`

and verify where your command definitions fits into.

```

cd itl/plugins-contrib.d/

ls

```

If you want to add or modify an existing Monitoring Plugin please use `itl/command-plugins.conf` instead.

```

vim itl/command-plugins-conf

```

##### Existing Configuration File

Just edit it, and add your CheckCommand definition.

```

vim operating-system.conf

```

Proceed to the documentation.

##### New type for CheckCommand Definition

Create a new file with .conf suffix.

```

$ vim printer.conf

```

Add the file to `itl/CMakeLists.txt` in the FILES line in \*\*alpha-numeric order\*\*.

This ensures that the installation and packages properly include your newly created file.

```

vim CMakeLists.txt

-FILES ipmi.conf network-components.conf operating-system.conf virtualization.conf vmware.conf

+FILES ipmi.conf network-components.conf operating-system.conf printer.conf virtualization.conf vmware.conf

```

Add the newly created file to your git commit.

```

git add printer.conf

```

Do not commit it yet but finish with the documentation.

#### <a id="contributing-itl-checkcommands-docs"></a> Create CheckCommand Documentation

Edit the documentation file in the `doc/` directory. More details on documentation

updates can be found [here](CONTRIBUTING.md#contributing-documentation).

```

vim doc/10-icinga-template-library.md

```

The CheckCommand documentation should be located in the same chapter

similar to the configuration file you have just added/modified.

Create a section for your plugin, add a description and a table of parameters. Each parameter should have at least:

\* optional or required

\* description of its purpose

\* the default value, if any

Look at the existing documentation and "copy" the same style and layout.

#### <a id="contributing-itl-checkcommands-patch"></a> Send a Patch

Commit your changes which includes a descriptive commit message.

```

git commit -av

Add printer CheckCommand definition

Explain its purpose and possible enhancements/shortcomings.

refs #existingticketnumberifany

```

Push the branch to the remote origin and create a [pull request](https://help.github.com/articles/using-pull-requests/).

```

git push --set-upstream origin feature/itl-check-printer

hub pull-request

```

In case developers ask for changes during review, please add them

to the branch and push those changes.

## <a id="contributing-review"></a> Review

### <a id="contributing-pr-review"></a> Pull Request Review

This is only important for developers who will review pull requests. If you want to join

the development team, kindly contact us.

- Ensure that the style guide applies.

- Verify that the patch fixes a problem or linked issue, if any.

- Discuss new features with team members.

- Test the patch in your local dev environment.

If there are changes required, kindly ask for an updated patch.

Once the review is completed, merge the PR via GitHub.

#### <a id="contributing-pr-review-fixes"></a> Pull Request Review Fixes

In order to amend the commit message, fix conflicts or add missing changes, you can

add your changes to the PR.

A PR is just a pointer to a different Git repository and branch.

By default, pull requests allow to push into the repository of the PR creator.

Example for [#4956](https://github.com/Icinga/icinga2/pull/4956):

At the bottom it says "Add more commits by pushing to the bugfix/persistent-comments-are-not-persistent branch on TheFlyingCorpse/icinga2."

First off, add the remote repository as additional origin and fetch its content:

```

git remote add theflyingcorpse https://github.com/TheFlyingCorpse/icinga2

git fetch --all

```

Checkout the mentioned remote branch into a local branch (Note: `theflyingcorpse` is the name of the remote):

```

git checkout theflyingcorpse/bugfix/persistent-comments-are-not-persistent -b bugfix/persistent-comments-are-not-persistent

```

Rebase, amend, squash or add your own commits on top.

Once you are satisfied, push the changes to the remote `theflyingcorpse` and its branch `bugfix/persistent-comments-are-not-persistent`.

The syntax here is `git push <remote> <localbranch>:<remotebranch>`.

```

git push theflyingcorpse bugfix/persistent-comments-are-not-persistent:bugfix/persistent-comments-are-not-persistent

```

In case you've changed the commit history (rebase, amend, squash), you'll need to force push. Be careful, this can't be reverted!

```

git push -f theflyingcorpse bugfix/persistent-comments-are-not-persistent:bugfix/persistent-comments-are-not-persistent

```