# How to contribute

Third-party patches are essential for keeping Puppet great. We simply can't

access the huge number of platforms and myriad configurations for running

Puppet. We want to keep it as easy as possible to contribute changes that

get things working in your environment. There are a few guidelines that we

need contributors to follow so that we can have a chance of keeping on

top of things.

## Puppet Core vs Modules

New functionality is typically directed toward modules to provide a slimmer

Puppet Core, reducing its surface area, and to allow greater freedom for

module maintainers to ship releases at their own cadence, rather than

being held to the cadence of Puppet releases. With Puppet 4's "all in one"

packaging, a list of modules at specific versions will be packaged with the

core so that popular types and providers will still be available as part of

the "out of the box" experience.

Generally, new types and new OS-specific providers for existing types should

be added in modules. Exceptions would be things like new cross-OS providers

and updates to existing core types.

If you are unsure of whether your contribution should be implemented as a

module or part of Puppet Core, you may visit [#puppet-dev on slack](https://puppetcommunity.slack.com/), or ask on

the [puppet-dev mailing list](https://groups.google.com/forum/#!forum/puppet-dev) for advice.

## Getting Started

\* Make sure you have a [Jira account](https://tickets.puppetlabs.com).

\* Make sure you have a [GitHub account](https://github.com/join).

\* Submit a Jira ticket for your issue if one does not already exist.

\* Clearly describe the issue including steps to reproduce when it is a bug.

\* Make sure you fill in the earliest version that you know has the issue.

\* A ticket is not necessary for [trivial changes](https://puppet.com/community/trivial-patch-exemption-policy)

\* Fork the repository on GitHub.

## Making Changes

\* Create a topic branch from where you want to base your work.

\* This is usually the master branch.

\* Only target release branches if you are certain your fix must be on that

branch.

\* To quickly create a topic branch based on master, run `git checkout -b

fix/master/my\_contribution master`. Please avoid working directly on the

`master` branch.

\* Make commits of logical and atomic units.

\* Check for unnecessary whitespace with `git diff --check` before committing.

\* Make sure your commit messages are in the proper format. If the commit

addresses an issue filed in the

[Puppet Jira project](https://tickets.puppetlabs.com/browse/PUP), start

the first line of the commit with the issue number in parentheses.

```

(PUP-1234) Make the example in CONTRIBUTING imperative and concrete

Without this patch applied the example commit message in the CONTRIBUTING

document is not a concrete example. This is a problem because the

contributor is left to imagine what the commit message should look like

based on a description rather than an example. This patch fixes the

problem by making the example concrete and imperative.

The first line is a real-life imperative statement with a ticket number

from our issue tracker. The body describes the behavior without the patch,

why this is a problem, and how the patch fixes the problem when applied.

```

\* Make sure you have added the necessary tests for your changes.

\* For details on how to run tests, please see [the quickstart guide](https://github.com/puppetlabs/puppet/blob/master/docs/quickstart.md)

## Writing Translatable Code

We use [gettext tooling](https://github.com/puppetlabs/gettext-setup-gem) to

extract user-facing strings and pull in translations based on the user's locale

at runtime. In order for this tooling to work, all user-facing strings must be

wrapped in the `\_()` translation function, so they can be extracted into files

for the translators.

When adding user-facing strings to your work, follow these guidelines:

\* Use full sentences. Strings built up out of concatenated bits are hard to translate.

\* Use string formatting instead of interpolation. Use the hash format and give good names to the placeholder values that can be used by translators to understand the meaning of the formatted values.

For example: `\_('Creating new user %{name}.') % { name: user.name }`

\* Use `n\_()` for pluralization. (see gettext gem docs linked above for details)

It is the responsibility of contributors and code reviewers to ensure that all

user-facing strings are marked in new PRs before merging.

## Making Trivial Changes

For [changes of a trivial nature](https://puppet.com/community/trivial-patch-exemption-policy), it is not always necessary to create a new

ticket in Jira. In this case, it is appropriate to start the first line of a

commit with one of `(docs)`, `(maint)`, or `(packaging)` instead of a ticket

number.

If a Jira ticket exists for the documentation commit, you can include it

after the `(docs)` token.

```

(docs)(DOCUMENT-000) Add docs commit example to CONTRIBUTING

There is no example for contributing a documentation commit

to the Puppet repository. This is a problem because the contributor

is left to assume how a commit of this nature may appear.

The first line is a real-life imperative statement with '(docs)' in

place of what would have been the PUP project ticket number in a

non-documentation related commit. The body describes the nature of

the new documentation or comments added.

```

For commits that address trivial repository maintenance tasks or packaging

issues, start the first line of the commit with `(maint)` or `(packaging)`,

respectively.

## Submitting Changes

\* Sign the [Contributor License Agreement](https://cla.puppet.com).

\* Push your changes to a topic branch in your fork of the repository.

\* Submit a pull request to the repository in the puppetlabs organization.

\* Update the related Jira ticket to mark that you have submitted code and are ready

for it to be reviewed (Status: Ready for Merge).

\* The core team looks at pull requests on a regular basis in a weekly triage

meeting.

\* After feedback has been given we expect responses within two weeks. After two

weeks we may close the pull request if it isn't showing any activity.

## Revert Policy

By running tests in advance and by engaging with peer review for prospective

changes, your contributions have a high probability of becoming long lived

parts of the the project. After being merged, the code will run through a

series of testing pipelines on a large number of operating system

environments. These pipelines can reveal incompatibilities that are difficult

to detect in advance.

If the code change results in a test failure, we will make our best effort to

correct the error. If a fix cannot be determined and committed within 24 hours

of its discovery, the commit(s) responsible \_may\_ be reverted, at the

discretion of the committer and Puppet maintainers. This action would be taken

to help maintain passing states in our testing pipelines.

The original contributor will be notified of the revert in the Jira ticket

associated with the change. A reference to the test(s) and operating system(s)

that failed as a result of the code change will also be added to the Jira

ticket. This test(s) should be used to check future submissions of the code to

ensure the issue has been resolved.

### Summary

\* Changes resulting in test pipeline failures will be reverted if they cannot

be resolved within one business day.

## Additional Resources

\* [Puppet community guidelines](https://puppet.com/community/community-guidelines)

\* [Bug tracker (Jira)](https://tickets.puppetlabs.com)

\* [Contributor License Agreement](https://cla.puppet.com/)

\* [General GitHub documentation](https://help.github.com/)

\* [GitHub pull request documentation](https://help.github.com/articles/creating-a-pull-request/)

\* [puppet-dev mailing list](https://groups.google.com/forum/#!forum/puppet-dev)

\* [Puppet community slack](https://slack.puppet.com)