The Ansible Development Cycle

Ansible developers (including community contributors) add new features, fix bugs, and update code in many different repositories. The ansible/ansible repository contains the code for basic features and functions, such as copying module code to managed nodes. This code is also known as `ansible-core`. Other repositories contain plugins and modules that enable Ansible to execute specific tasks, like adding a user to a particular database or configuring a particular network device. These repositories contain the source code for collections.

Development on `ansible-core` occurs on two levels. At the macro level, the `ansible-core` developers and maintainers plan releases and track progress with roadmaps and projects. At the micro level, each PR has its own lifecycle.

Development on collections also occurs at the macro and micro levels. Each collection has its own macro development cycle. For more information on the collections development cycle, see :ref:`contributing_maintained_collections`. The micro-level lifecycle of a PR is similar in collections and in `ansible-core`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(rst)(community)development_process.rst`, **line 11**); *backlink*
>
> Unknown interpreted text role "ref".

Macro development: `ansible-core` roadmaps, releases, and projects

If you want to follow the conversation about what features will be added to `ansible-core` for upcoming releases and what bugs are being fixed, you can watch these resources:

- the :ref:`roadmaps`

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst`, **line 21**); *backlink*
  >
  > Unknown interpreted text role "ref".

- the :ref:`Ansible Release Schedule <release_and_maintenance>`

  > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst`, **line 22**); *backlink*
  >
  > Unknown interpreted text role "ref".

- various GitHub projects - for example:
  - the 2.12 release project
  - the core documentation project

Micro development: the lifecycle of a PR

If you want to contribute a feature or fix a bug in `ansible-core` or in a collection, you must open a **pull request** ("PR" for short). GitHub provides a great overview of how the pull request process works in general. The ultimate goal of any pull request is to get merged and become part of a collection or `ansible-core`. Here's an overview of the PR lifecycle:

- Contributor opens a PR
- Ansibot reviews the PR
- Ansibot assigns labels
- Ansibot pings maintainers
- Azure Pipelines runs the test suite
- Developers, maintainers, community review the PR
- Contributor addresses any feedback from reviewers
- Developers, maintainers, community re-review
- PR merged or closed

Automated PR review: ansibullbot

Because Ansible receives many pull requests, and because we love automating things, we have automated several steps of the process of reviewing and merging pull requests with a tool called Ansibullbot, or Ansibot for short.

Ansibullbot serves many functions:

- Responds quickly to PR submitters to thank them for submitting their PR
- Identifies the community maintainer responsible for reviewing PRs for any files affected
- Tracks the current status of PRs
- Pings responsible parties to remind them of any PR actions for which they may be responsible
- Provides maintainers with the ability to move PRs through the workflow
- Identifies PRs abandoned by their submitters so that we can close them
- Identifies modules abandoned by their maintainers so that we can find new maintainers

Ansibot workflow

Ansibullbot runs continuously. You can generally expect to see changes to your issue or pull request within thirty minutes. Ansibullbot examines every open pull request in the repositories, and enforces state roughly according to the following workflow:

- If a pull request has no workflow labels, it's considered **new**. Files in the pull request are identified, and the maintainers of those files are pinged by the bot, along with instructions on how to review the pull request. (Note: sometimes we strip labels from a pull request to "reboot" this process.)
- If the module maintainer is not `$team_ansible`, the pull request then goes into the **community_review** state.
- If the module maintainer is `$team_ansible`, the pull request then goes into the **core_review** state (and probably sits for a while).
- If the pull request is in **community_review** and has received comments from the maintainer:
  - If the maintainer says `shipit`, the pull request is labeled **shipit**, whereupon the Core team assesses it for final merge.
  - If the maintainer says `needs_info`, the pull request is labeled **needs_info** and the submitter is asked for more info.

    - If the maintainer says **needs_revision**, the pull request is labeled **needs_revision** and the submitter is asked to fix some things.
- If the submitter says `ready_for_review`, the pull request is put back into **community_review** or **core_review** and the maintainer is notified that the pull request is ready to be reviewed again.
- If the pull request is labeled **needs_revision** or **needs_info** and the submitter has not responded lately:
    - The submitter is first politely pinged after two weeks, pinged again after two more weeks and labeled **pending action**, and the issue or pull request will be closed two weeks after that.
    - If the submitter responds at all, the clock is reset.
- If the pull request is labeled **community_review** and the reviewer has not responded lately:
    - The reviewer is first politely pinged after two weeks, pinged again after two more weeks and labeled **pending_action**, and then may be reassigned to `$team_ansible` or labeled **core_review**, or often the submitter of the pull request is asked to step up as a maintainer.
- If Azure Pipelines tests fail, or if the code is not able to be merged, the pull request is automatically put into **needs_revision** along with a message to the submitter explaining why.

There are corner cases and frequent refinements, but this is the workflow in general.

PR labels

There are two types of PR Labels generally: **workflow** labels and **information** labels.

Workflow labels

- **community_review**: Pull requests for modules that are currently awaiting review by their maintainers in the Ansible community.
- **core_review**: Pull requests for modules that are currently awaiting review by their maintainers on the Ansible Core team.
- **needs_info**: Waiting on info from the submitter.
- **needs_rebase**: Waiting on the submitter to rebase.
- **needs_revision**: Waiting on the submitter to make changes.
- **shipit**: Waiting for final review by the core team for potential merge.

Information labels

- **backport**: this is applied automatically if the PR is requested against any branch that is not devel. The bot immediately assigns the labels backport and `core_review`.
- **bugfix_pull_request**: applied by the bot based on the templatized description of the PR.
- **cloud**: applied by the bot based on the paths of the modified files.
- **docs_pull_request**: applied by the bot based on the templatized description of the PR.
- **easyfix**: applied manually, inconsistently used but sometimes useful.
- **feature_pull_request**: applied by the bot based on the templatized description of the PR.
- **networking**: applied by the bot based on the paths of the modified files.
- **owner_pr**: largely deprecated. Formerly workflow, now informational. Originally, PRs submitted by the maintainer would automatically go to **shipit** based on this label. If the submitter is also a maintainer, we notify the other maintainers and still require one of the maintainers (including the submitter) to give a **shipit**.
- **pending_action**: applied by the bot to PRs that are not moving. Reviewed every couple of weeks by the community team, who tries to figure out the appropriate action (closure, asking for new maintainers, and so on).

Special Labels

- **new_plugin**: this is for new modules or plugins that are not yet in Ansible.

Note: *new_plugin* kicks off a completely separate process, and frankly it doesn't work very well at present. We're working our best to improve this process.

Human PR review

After Ansibot reviews the PR and applies labels, the PR is ready for human review. The most likely reviewers for any PR are the maintainers for the module that PR modifies.

Each module has at least one assigned :ref:`maintainer <maintainers>`, listed in the BOTMETA.yml file.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst`, **line 129**); *backlink*
>
> Unknown interpreted text role "ref".

The maintainer's job is to review PRs that affect that module and decide whether they should be merged (`shipit`) or revised (`needs_revision`). We'd like to have at least one community maintainer for every module. If a module has no community maintainers assigned, the maintainer is listed as `$team_ansible`.

Once a human applies the `shipit` label, the :ref:`committers <community_committer_guidelines>` decide whether the PR is ready to be merged. Not every PR that gets the `shipit` label is actually ready to be merged, but the better our reviewers are, and the better our guidelines are, the more likely it will be that a PR that reaches **shipit** will be mergeable.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst`, **line 133**); *backlink*
>
> Unknown interpreted text role "ref".

Making your PR merge-worthy

We do not merge every PR. Here are some tips for making your PR useful, attractive, and merge-worthy.

Creating changelog fragments

Changelogs help users and developers keep up with changes to ansible-core and Ansible collections. Ansible and many collections build changelogs for each release from fragments. For ansible-core and collections using this model, you **must** add a changelog fragment to any PR that changes functionality or fixes a bug.

You do not need a changelog fragment for PRs that:

- add new modules and plugins, because Ansible tooling does that automatically;
- contain only documentation changes.

> **Note**
>
> Some collections require a changelog fragment for every pull request. They use the `trivial:` section for entries mentioned above that will be skipped when building a release changelog.

More precisely:

- Every bugfix PR must have a changelog fragment. The only exception are fixes to a change that has not yet been included in a release.
- Every feature PR must have a changelog fragment.
- New modules and plugins (except jinja2 filter and test plugins) must have `versions_added` set correctly, and do not need a changelog fragment. The tooling detects new modules and plugins by their `versions_added` value and announces them in the next release's changelog automatically.
- New jinja2 filter and test plugins, and also new roles and playbooks (for collections) must have a changelog fragment. See :ref:`changelogs_how_to_format_j2_roles_playbooks` or the antsibull-changelog documentation for such changelog fragments

for information on what the fragments should look like.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 162**); *backlink*
>
> Unknown interpreted text role "ref".

We build short summary changelogs for minor releases as well as for major releases. If you backport a bugfix, include a changelog fragment with the backport PR.

Creating a changelog fragment

A basic changelog fragment is a `.yaml` or `.yml` file placed in the `changelogs/fragments/` directory. Each file contains a yaml dict with keys like `bugfixes` or `major_changes` followed by a list of changelog entries of bugfixes or features. Each changelog entry is rst embedded inside of the yaml file which means that certain constructs would need to be escaped so they can be interpreted by rst and not by yaml (or escaped for both yaml and rst if you prefer). Each PR **must** use a new fragment file rather than adding to an existing one, so we can trace the change back to the PR that introduced it.

PRs which add a new module or plugin do not necessarily need a changelog fragment. See the previous section :ref:`community_changelogs`. Also see the next section :ref:`changelogs_how_to_format` for the precise format changelog fragments should have.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 173**); *backlink*
>
> Unknown interpreted text role "ref".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 173**); *backlink*
>
> Unknown interpreted text role "ref".

To create a changelog entry, create a new file with a unique name in the `changelogs/fragments/` directory of the corresponding repository. The file name should include the PR number and a description of the change. It must end with the file extension `.yaml` or `.yml`. For example: `40696-user-backup-shadow-file.yaml`

A single changelog fragment may contain multiple sections but most will only contain one section. The toplevel keys (bugfixes, major_changes, and so on) are defined in the config file for our release note tool. Here are the valid sections and a description of each:

breaking_changes

MUST include changes that break existing playbooks or roles. This includes any change to existing behavior that forces users to update tasks. Breaking changes means the user MUST make a change when they update. Breaking changes MUST only happen in a major release of the collection. Write in present tense and clearly describe the new behavior that the end user must now follow. Displayed in both the changelogs and the :ref:`Porting Guides <porting_guides>`.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 180**); *backlink*
>
> Unknown interpreted text role "ref".

```
breaking_changes:
  - ansible-test - automatic installation of requirements for cloud test plugins no longer occurs. The affected test plugins are
```

major_changes

Major changes to ansible-core or a collection. SHOULD NOT include individual module or plugin changes. MUST include non-breaking changes that impact all or most of a collection (for example, updates to support a new SDK version across the collection). Major changes mean the user can CHOOSE to make a change when they update but do not have to. Could be used to announce an important upcoming EOL or breaking change in a future release. (ideally 6 months in advance, if known. See this example). Write in present tense and describe what is new. Optionally, include a 'Previously..." sentence to help the user identify where old behavior should now change. Displayed in both the changelogs and the :ref:`Porting Guides <porting_guides>`.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 189**); *backlink*
>
> Unknown interpreted text role "ref".

```
major_changes:
  - ansible-test - all cloud plugins which use containers can now be used with all POSIX and Windows hosts. Previously the plugin
```

minor_changes

Minor changes to ansible-core, modules, or plugins. This includes new parameters added to modules, or non-breaking behavior changes to existing parameters, such as adding additional values to choices[]. Minor changes are enhancements, not bug fixes. Write in present tense.

```
minor_changes:
  - lineinfile - add warning when using an empty regexp (https://github.com/ansible/ansible/issues/29443).
```

deprecated_features

Features that have been deprecated and are scheduled for removal in a future release. Use past tense and include an alternative, where available for what is being deprecated.. Displayed in both the changelogs and the :ref:`Porting Guides <porting_guides>`.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 206**); *backlink*
>
> Unknown interpreted text role "ref".

```
deprecated features:
  - include action - is deprecated in favor of ``include_tasks``, ``import_tasks`` and ``import_playbook`` (https://github.com/an
```

removed_features

Features that were previously deprecated and are now removed. Use past tense and include an alternative, where available for what is being deprecated. Displayed in both the changelogs and the :ref:`Porting Guides <porting_guides>`.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst, **line 215**); *backlink*
>
> Unknown interpreted text role "ref".

```
removed_features:
  - _get_item() alias - removed from callback plugin base class which had been deprecated in favor of ``_get_item_label()`` (http
```

security_fixes

Fixes that address CVEs or resolve security concerns. MUST use security_fixes for any CVEs. Use present tense. Include links to CVE information.

```
security_fixes:
  - set_options -do not include params in exception when a call to ``set_options`` fails. Additionally, block the exception that
```

bugfixes

Fixes that resolve issues. SHOULD not be used for minor enhancements (use `minor_change` instead). Use past tense to describe the problem and present tense to describe the fix.

```
bugfixes:
  - ansible_play_batch - variable included unreachable hosts. Fix now saves unreachable hosts between plays by adding them to the
```

known_issues

Known issues that are currently not fixed or will not be fixed. Use present tense and where available, use imperative tense for a workaround.

```
known_issues:
  - ansible-test - tab completion anywhere other than the end of the command with the new composite options provides incorrect re
```

Each changelog entry must contain a link to its issue between parentheses at the end. If there is no corresponding issue, the entry must contain a link to the PR itself.

Most changelog entries are `bugfixes` or `minor_changes`. The changelog tool also supports `trivial`, which are not listed in the actual changelog output but are used by collections repositories that require a changelog fragment for each PR.

Changelog fragment entry format

When writing a changelog entry, use the following format:

```
- scope - description starting with a lowercase letter and ending with a period at the very end. Multiple sentences are allowed (https:/
```

The scope is usually a module or plugin name or group of modules or plugins, for example, `lookup plugins`. While module names can (and should) be mentioned directly (`foo_module`), plugin names should always be followed by the type (`foo inventory plugin`).

For changes that are not really scoped (for example, which affect a whole collection), use the following format:

```
- Description starting with an uppercase letter and ending with a dot at the very end. Multiple sentences are allowed (https://github.co
```

Here are some examples:

```
bugfixes:
  - apt_repository - fix crash caused by ``cache.update()`` raising an ``IOError``
    due to a timeout in ``apt update`` (https://github.com/ansible/ansible/issues/51995).
```

```
minor_changes:
  - lineinfile - add warning when using an empty regexp (https://github.com/ansible/ansible/issues/29443).
```

```
bugfixes:
  - copy - the module was attempting to change the mode of files for
    remote_src=True even if mode was not set as a parameter.  This failed on
    filesystems which do not have permission bits (https://github.com/ansible/ansible/issues/29444).
```

You can find more example changelog fragments in the changelog directory for the 2.12 release.

After you have written the changelog fragment for your PR, commit the file and include it with the pull request.

Changelog fragment entry format for new jinja2 plugins, roles, and playbooks

While new modules and plugins that are not jinja2 filter or test plugins are mentioned automatically in the generated changelog, jinja2 filter and test plugins, roles, and playbooks are not. To make sure they are mentioned, a changelog fragment in a specific format is needed:

```
# A new jinja2 filter plugin:
add plugin.filter:
  - # The following needs to be the name of the filter itself, not of the file
    # the filter is included in!
    name: to_time_unit
    # The description should be in the same format as short_description for
    # other plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: Converts a time expression to a given unit

# A new jinja2 test plugin:
add plugin.test:
  - # The following needs to be the name of the test itself, not of the file
    # the test is included in!
    name: asn1time
    # The description should be in the same format as short_description for
    # other plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: Check whether the given string is an ASN.1 time

# A new role:
add object.role:
  - # This should be the short (non-FQCN) name of the role.
    name: nginx
    # The description should be in the same format as short_description for
    # plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: A nginx installation role

# A new playbook:
add object.playbook:
  - # This should be the short (non-FQCN) name of the playbook.
    name: wipe server
    # The description should be in the same format as short description for
    # plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: Wipes a server
```

Backporting merged PRs in `ansible-core`

All `ansible-core` PRs must be merged to the `devel` branch first. After a pull request has been accepted and merged to the `devel` branch, the following instructions will help you create a pull request to backport the change to a previous stable branch.

We do **not** backport features.

> **Note**
>
> These instructions assume that:
>
> - `stable-2.12` is the targeted release branch for the backport
> - `https://github.com/ansible/ansible.git` is configured as a `git remote` named `upstream`. If you do not use a `git remote` named `upstream`, adjust the instructions accordingly.
> - `https://github.com/<yourgithubaccount>/ansible.git` is configured as a `git remote` named `origin`. If you do not use a `git remote` named `origin`, adjust the instructions

accordingly.

1. Prepare your devel, stable, and feature branches:

```
git fetch upstream
git checkout -b backport/2.12/[PR_NUMBER_FROM_DEVEL] upstream/stable-2.12
```

1. Cherry pick the relevant commit SHA from the devel branch into your feature branch, handling merge conflicts as necessary:

```
git cherry-pick -x [SHA_FROM_DEVEL]
```

1. Add a :ref:`changelog fragment <changelogs_how_to>` for the change, and commit it.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\(ansible-devel)(docs)(docsite)(rst)(community)development_process.rst`, line 377); *backlink*
>
> Unknown interpreted text role "ref".

2. Push your feature branch to your fork on GitHub:

```
git push origin backport/2.12/[PR_NUMBER_FROM_DEVEL]
```

1. Submit the pull request for `backport/2.12/[PR_NUMBER_FROM_DEVEL]` against the `stable-2.12` branch
2. The Release Manager will decide whether to merge the backport PR before the next minor release. There isn't any need to follow up. Just ensure that the automated tests (CI) are green.

> **Note**
>
> The branch name `backport/2.12/[PR_NUMBER_FROM_DEVEL]` is somewhat arbitrary, but conveys meaning about the purpose of the branch. This branch name format is not required, but it can be helpful, especially when making multiple backport PRs for multiple stable branches.

> **Note**
>
> If you prefer, you can use CPython's cherry-picker tool (`pip install --user 'cherry-picker >= 1.3.2'`) to backport commits from devel to stable branches in Ansible. Take a look at the cherry-picker documentation for details on installing, configuring, and using it.