DEVELOPMENT WORKFLOWS

This page explains the workflows a developer is expected to follow to implement the goals that are part of the Ceph release cycle. It does not go into technical details and is designed to provide a high level view instead. Each chapter is about a given goal such as `Merging bug fixes or features` or `Publishing point releases and backporting`.

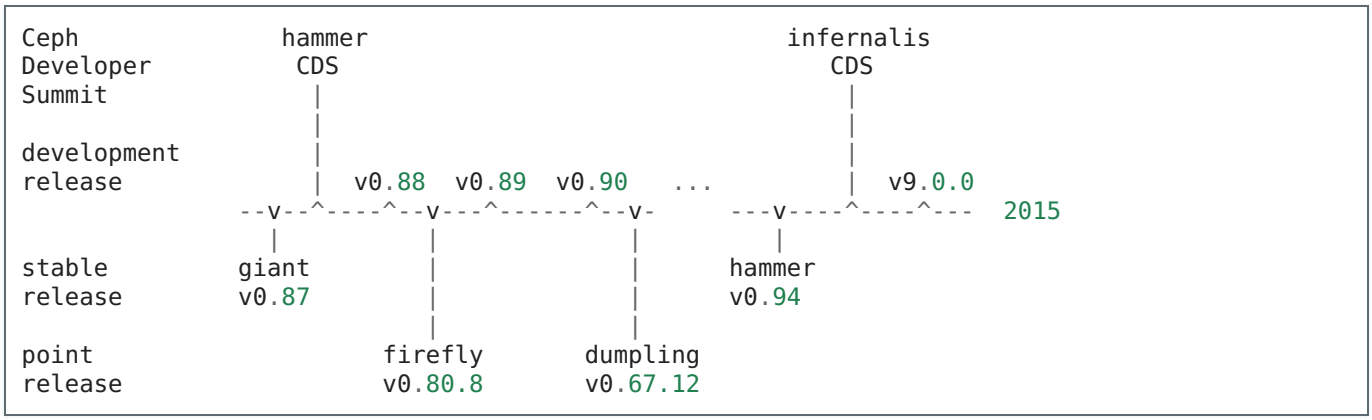
A key aspect of all workflows is that none of them blocks another. For instance, a bug fix can be backported and merged to a stable branch while the next point release is being published. For that specific example to work, a branch should be created to avoid any interference. In practice it is not necessary for Ceph because:

- there are few people involved
- the frequency of backports is not too high
- the reviewers, who know a release is being published, are unlikely to merge anything that may cause issues

This ad-hoc approach implies the workflows are changed on a regular basis to adapt. For instance, `quality engineers` were not involved in the workflow to publish `dumpling` point releases. The number of commits being backported to `firefly` made it impractical for developers tasked to write code or fix bugs to also run and verify the full suite of integration tests. Inserting `quality engineers` makes it possible for someone to participate in the workflow by analyzing test results.

The workflows are not enforced when they impose an overhead that does not make sense. For instance, if the release notes for a point release were not written prior to checking all integration tests, they can be commited to the stable branch and the result sent for publication without going through another run of integration tests.

RELEASE CYCLE



Four times a year, the development roadmap is discussed online during the Ceph Developer Summit. A new stable release (hammer, infernalis, jewel ...) is published at the same frequency. Every other release (firefly, hammer, jewel...) is a Long Term Stable (LTS). See Understanding the release cycle for more information.

MERGING BUG FIXES OR FEATURES

The development branch is `master` and the workflow followed by all developers can be summarized as follows:

- The developer prepares a series of commits
- The developer submits the series of commits via a pull request
- A reviewer is assigned the pull request
- When the pull request looks good to the reviewer, it is merged into an integration branch by the tester
- After a successful run of integration tests, the pull request is merged by the tester

The `developer` is the author of a series of commits. The `reviewer` is responsible for providing feedback to the developer on a regular basis and the developer is invited to ping the reviewer if nothing happened after a week. After the `reviewer` is satisfied with the pull request, (s)he passes it to the `tester`. The `tester` is responsible for running teuthology integration tests on the pull request. If nothing happens within a month the `reviewer` is invited to ping the `tester`.

RESOLVING BUG REPORTS AND IMPLEMENTING FEATURES

All bug reports and feature requests are in the issue tracker and the workflow can be summarized as follows:

- The reporter creates the issue with priority `Normal`
- A developer may pick the issue right away
- During a bi-weekly bug scrub, the team goes over all new issue and assign them a priority
- The bugs with higher priority are worked on first

Each `team` is responsible for a project, managed by leads.

The `developer` assigned to an issue is responsible for it. The status of an open issue can be:

- New: it is unclear if the issue needs work.
- `Verified`: the bug can be reproduced or showed up multiple times
- `In Progress`: the developer is working on it this week
- `Pending Backport`: the fix needs to be backported to the stable releases listed in the backport field

For each `Pending Backport` issue, there exists at least one issue in the `Backport` tracker to record the work done to cherry pick the necessary commits from the master branch to the target stable branch. See the backporter manual for more information.

RUNNING AND INTERPRETING TEUTHOLOGY INTEGRATION TESTS

The Sepia community test lab runs teuthology integration tests on a regular basis and the results are posted on pulpito and the ceph-qa mailing list.

- The job failures are analyzed by quality engineers and developers
- If the cause is environmental (e.g. network connectivity), an issue is created in the sepia lab project
- If the bug is known, a pulpito URL to the failed job is added to the issue
- If the bug is new, an issue is created

The `quality engineer` is either a developer or a member of the QE team. There is at least one integration test suite per project:

- rgw suite
- CephFS suite
- rados suite
- rbd suite

and many others such as

- upgrade suites
- power-cyle suite
- ...

PREPARING A NEW RELEASE

A release is prepared in a dedicated branch, different from the `master` branch.

- For a stable releases it is the branch matching the release code name (dumpling, firefly, etc.)
- For a development release it is the `next` branch

The workflow expected of all developers to stabilize the release candidate is the same as the normal development workflow with the following differences:

- The pull requests must target the stable branch or next instead of master
- The reviewer rejects pull requests that are not bug fixes
- The `Backport` issues matching a teuthology test failure and set with priority `Urgent` must be fixed before the release

CUTTING A NEW STABLE RELEASE

A new stable release can be cut when:

- all `Backport` issues with priority `Urgent` are fixed
- integration and upgrade tests run successfully

Publishing a new stable release implies a risk of regression or discovering new bugs during the upgrade, no matter how carefully it is tested. The decision to cut a release must take this into account: it may not be wise to publish a stable release that only fixes a few minor bugs. For instance if only one commit has been backported to a stable release that is not a LTS, it is

better to wait until there are more.

When a stable release is to be retired, it may be safer to recommend an upgrade to the next LTS release instead of proposing a new point release to fix a problem. For instance, the `dumpling` v0.67.11 release has bugs related to backfilling which have been fixed in `firefly` v0.80.x. A backport fixing these backfilling bugs has been tested in the draft point release `dumpling` v0.67.12 but they are large enough to introduce a risk of regression. As `dumpling` is to be retired, users suffering from this bug can upgrade to `firefly` to fix it. Unless users manifest themselves and ask for `dumpling` v0.67.12, this draft release may never be published.

- The `Ceph lead` decides a new stable release must be published
- The `release master` gets approval from all leads
- The `release master` writes and commits the release notes
- The `release master` informs the `quality engineer` that the branch is ready for testing
- The `quality engineer` runs additional integration tests
- If the `quality engineer` discovers new bugs that require an `Urgent Backport`, the release goes back to being prepared, it was not ready after all
- The `quality engineer` informs the `publisher` that the branch is ready for release
- The `publisher` creates the packages and sets the release tag

The person responsible for each role is:

- Sage Weil is the `Ceph lead`
- Sage Weil is the `release master` for major stable releases (`firefly` 0.80, `hammer` 0.94 etc.)
- Loic Dachary is the `release master` for stable point releases (`firefly` 0.80.10, `hammer` 0.94.1 etc.)
- Yuri Weinstein is the `quality engineer`
- Alfredo Deza is the `publisher`

CUTTING A NEW DEVELOPMENT RELEASE

The publication workflow of a development release is the same as preparing a new release and cutting it, with the following differences:

- The `next` branch is reset to the tip of `master` after publication
- The `quality engineer` is not required to run additional tests, the `release master` directly informs the `publisher` that the release is ready to be published.

PUBLISHING POINT RELEASES AND BACKPORTING

The publication workflow of the point releases is the same as preparing a new release and cutting it, with the following differences:

- The `backport` field of each issue contains the code name of the stable release
- There is exactly one issue in the `Backport` tracker for each stable release to which the issue is backported
- All commits are cherry-picked with `git cherry-pick -x` to reference the original commit

See the backporter manual for more information.