Release principles for Apache CloudStack 4.6 and up

This page describes the Release Management principles for Apache CloudStack 4.6 and newer releases.

Contents:

TL;DR The release principles

1. Master needs to be stable at all times
2. Pull requests will be merged after 2x LGTM and no -1 (see below)
3. When a release is being prepared, master will be “frozen” for new features (we aim to keep this window as short as possible)
4. Release branch will be branched off of master as soon as a release candidate (RC) vote passes (no more QA on release branches before release)
5. Bug fixes should be fixed on a release branch first, then merged forward to the next release (if any) and finally to master.
   Important: The commit hashes from the Pull Request should stil be the same in all branches this commit is in (cherry-picking cannot do this).
6. Only bug fixes will be fixed in release branches, there will be no back porting of new features
7. We should all use the same scripts to merge pull requests and do forward merges. The tools are located in the CloudStack repository.

Goal written in “Scrum”-style story

```
As a RM I want master to be stable at all times
```

so that I can create release candidates of high quality

that require little QA and can thus be released fast and often.

Master needs to be stable at all times

Stable master means all code can be cleanly compiled, all automated tests are passing (giving enough room for exceptions when automated test are flakey), and test coverage does not go down* (otherwise that would render the automated testing less useful every time).

*Coverage may go down if code that was covered is deleted

About LGTM

LGTM, "*Looks Good To Me*" is given once a reviewer of a Pull Requests gives an OK to proceed.

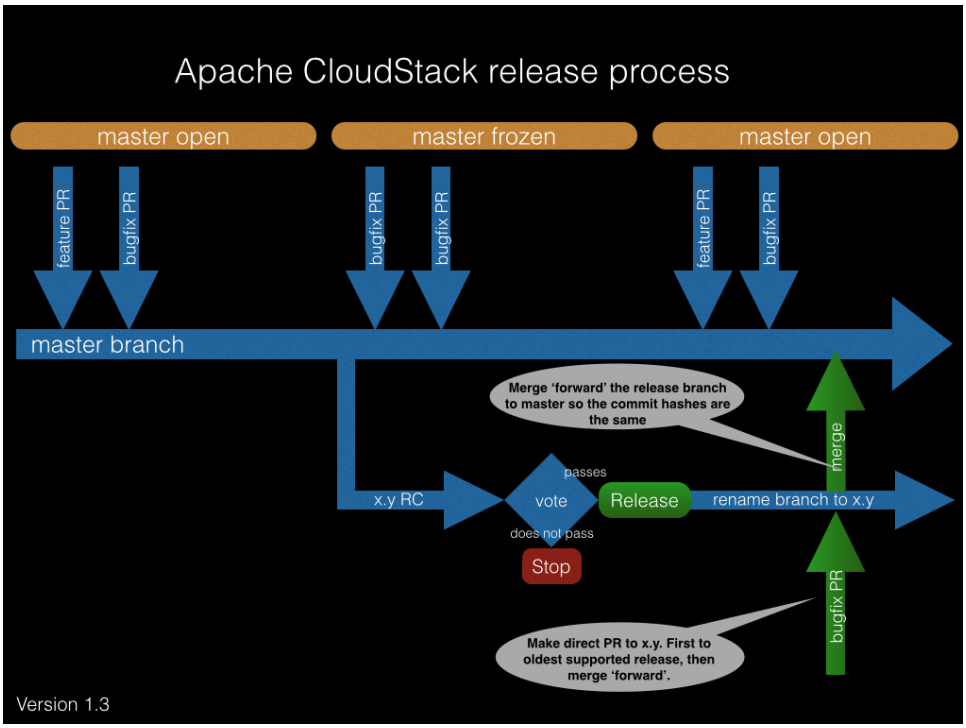
Please note:

- At least one of the reviews needs to run the Marvin integration tests and post output of a successful run. This is to prevent regression. Another review can then focus on the code itself, for example.
  Should running the Marvin tests make no sense (for example when the Pull Request only changes the UI), the reviewer should post other "proof" that it worked for him, like a screenshot
- Any LGTM without details on what the reviewer did, will not be considered in the LGTM count
- Before merging, any open questions and comments should be addressed
- Pull Requests that fail the above requirements and are merged anyway, will be reverted

Overview of release process

The release process would work like this (x=major, y=minor):

- start to branch RC off from master for the new upcoming version
- when not voted in, abandon RC branch, make fixes on master and start release process from the beginning
- when voted in, x.y.0 is released and the RC-branch will become x.y branch and the commit that has been voted on, i.e. the head of branch x.y, tagged x.y.0



Preparing new release: master frozen

- When the x.y release is being prepared, master will be “frozen” for new features. The release managers decide what goes in and what not.
- Working on new features will continue in feature branches, that can be easily rebased against master since it is stable and only receives bug fixes. The easiest would be to work on a feature branch in your own fork, although committers may choose to maintain a feature branch at the Apache CloudStack git repo instead.
- The time master is frozen should be as short as possible. It would be awesome if we could reach a schedule where master is open the first half of a month, and frozen the second half. Or even open for 12 days and then frozen for a weekend. But let's first try this and see how it works. We'll be able to speed it up as we go.

After x.y.0 release, master is open again

- Once x.y.0 is released, master will be opened for new feature merges. The Release Managers announce this on the list (usually a few days are needed to finalise the release and prepare the branches for the next release).
- When the new features are in, master will be frozen again, etc.
- It is expected that the new version can go out soon, as we start off from an already stable master at the exact point the previous release was made.

Release branch

- A release branch is created only at RC-vote time. When the vote passes, the branch is kept to be able to release upcoming point releases from it (x.y.1, x.y.2, etc)
- We will only fix bugs in this release branch, no back porting of new features

Bug fixes

- It's very important that we can easily track bug fix commits, so their hashes should remain the same in all branches
- Therefore, a pull request (PR) that fixes a bug, should be sent against a release branch
- This can be either the "current release" or the "previous release", depending on which ones are maintained

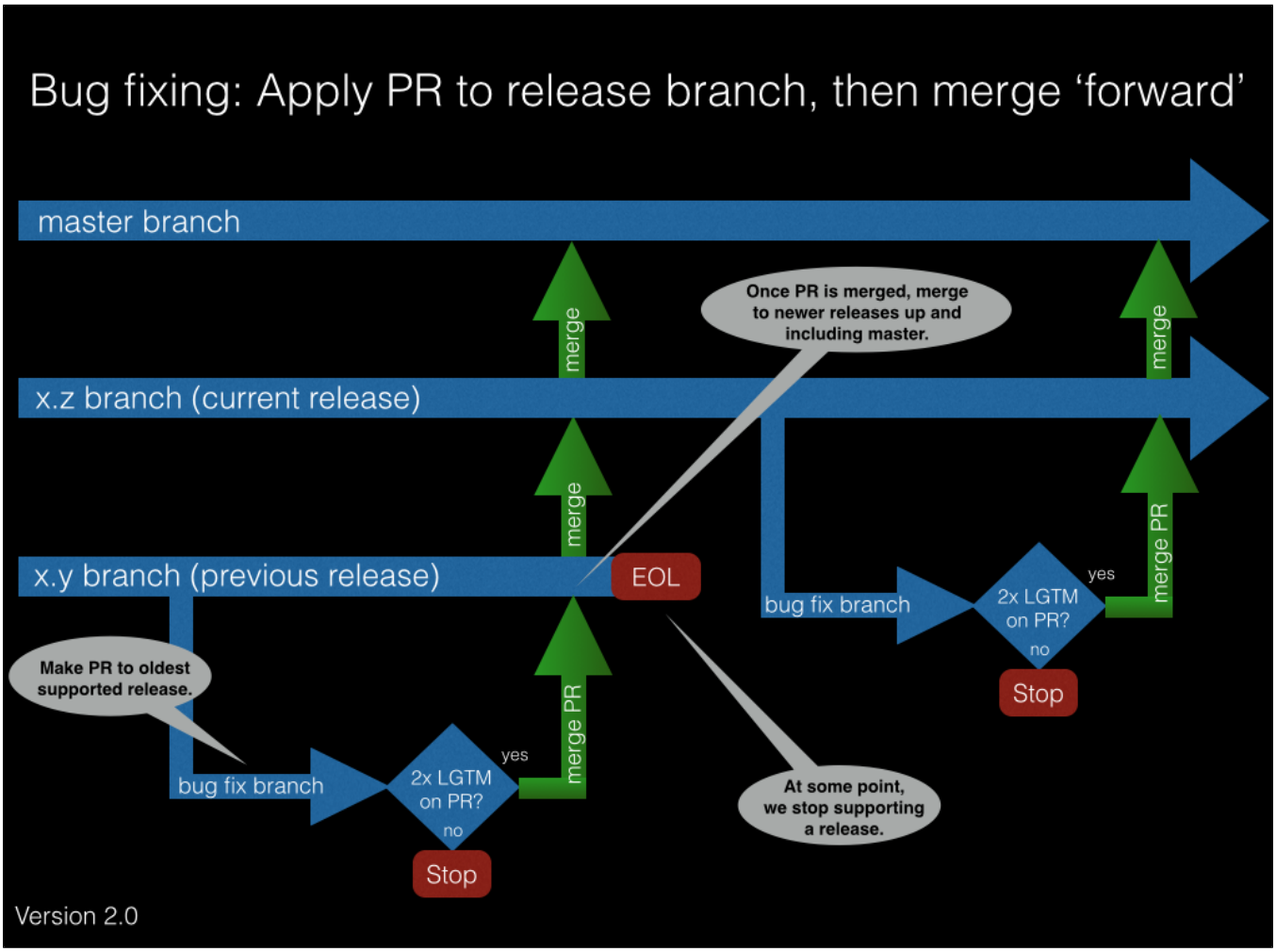
- Since the goal is a stable master, bug fixes should be "merged forward" to the next branch in order: "previous release" -> "current release" -> master (in other words: old to new)
- When we get a PR that fixes a bug that is not against a release branch, we should ask the author if the bug is also present in release branches. If so, the PR should be against the release branch and then be merged forward to master.
- Not the other way around, as that would involve cherry-picking which generates new commit hashes.

Important

⚠

Starting in 4.6, we **no longer use cherry-picking** to get commits from master to release branches.

Instead, we merge a PR to a release branch, then **merge it forward** to the next release and finally master. See image for example.

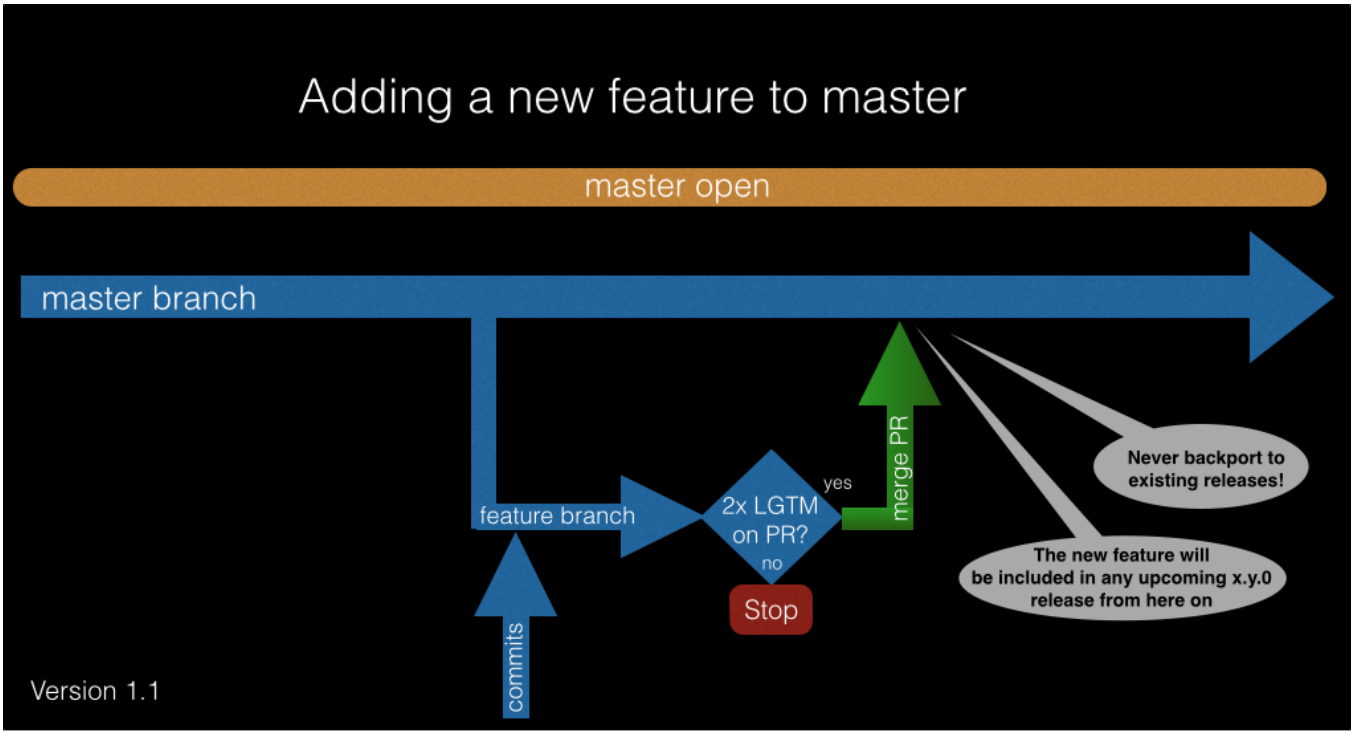


Which releases to maintain?

- Ideally, we maintain master and the latest release branch.
- Upgrades should be quick and painless so people can keep close to the latest release. Until we are there we will make sure to support our users as good as we can. We will put effort in upgrades with respect to proving these 'simply work ™'.

Developing new features

- Development should be done in a feature branch, branched off of master
- Send a PR to get it into master (2x LGTM applies), represented by the 'merge PR' arrows below
- Will only be merged when master is open, will be held otherwise until master is open again
- No back porting / cherry-picking features to existing branches!



How to merge a Pull Request?

See this wiki article for a guide on how to do it.

Why merging forward over cherry-picking from master?

The biggest discussion we had when writing up these principles, was the choice between either:

1. merge bug fix pull requests to master, then cherry-pick them to other releases
2. merge bug fix pull requests to the oldest supported release, then merge forward to the next release until we merge to master

The benefit of 1) is that we know everything will be in master. The downside is that cherry-picking changes the commit hashes which makes it hard to find in which releases a given commit is.

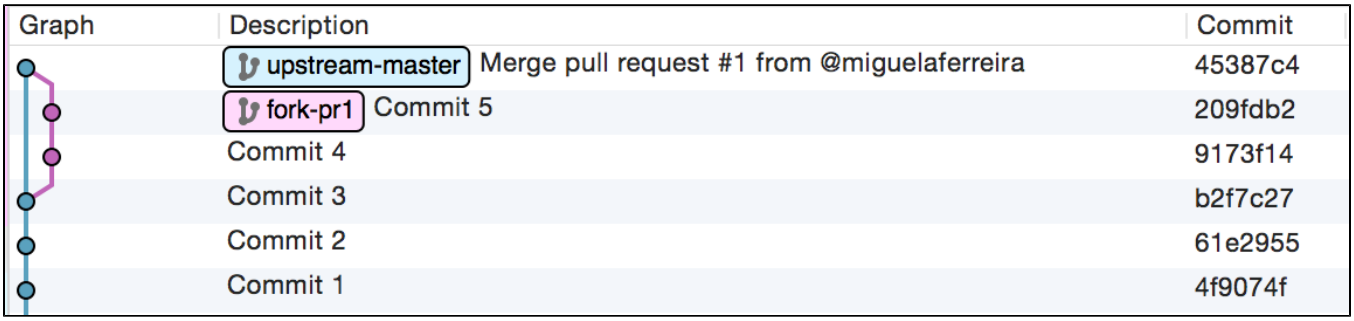
The big benefit of 2) is that when we merge a pull request to the oldest release and merge it forward, the commit hashes stay the same all the time. We also automatically make sure that bugs are fixes in supported releases.

As a bonus of 2), when we work like this, GitHub will much better detect we merged certain pull requests that now stay "open", even when we write "This closes #1234.

That's why we decided to go for option number 2.

Scenario 1: Merging a pull request to master

wiki with detailed info here



Git commands:

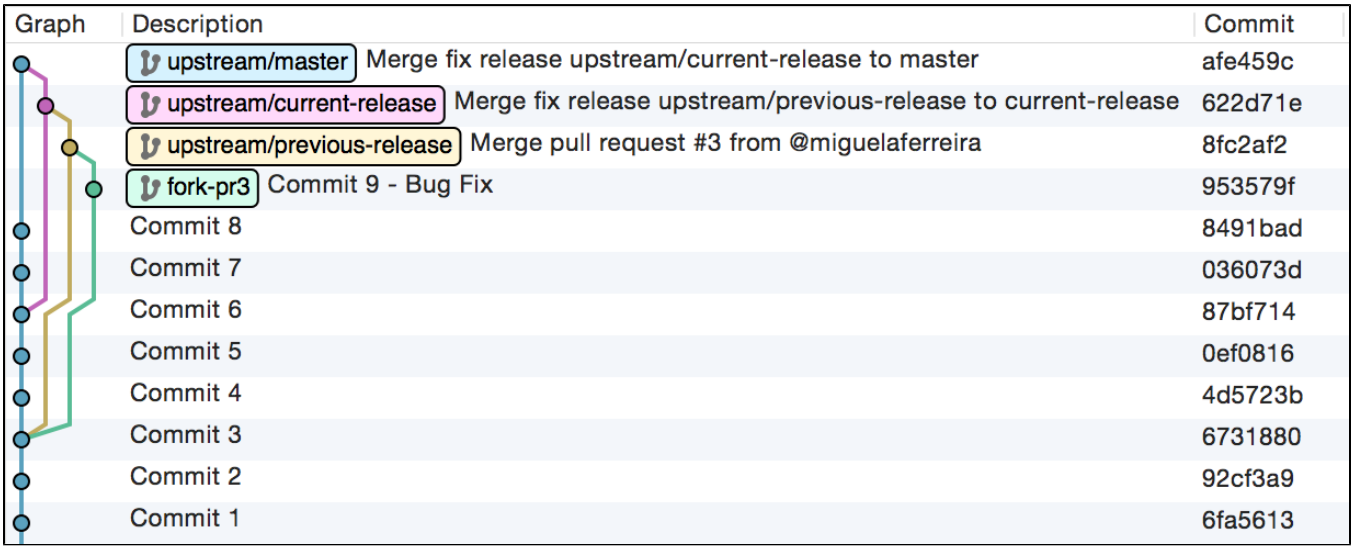
Merge pull request

```
# Get the exact same commits from the pull-request (the commit hashes must not change)
git fetch origin pull/${prId}/head:pr/${prId}

# Commit it without fast-forwarding. Type the commit message as detailed below.
git merge --no-ff --log pr/${prId}


# Finally, sign the merge commit.
git commit --amend -s --allow-empty-message -m ''
```

An optional helper script with detailed usage and help can be found here.

Scenario 2: Merging a pull request to a release branch

wiki with detailed info here



Forward merge

```
# Make sure you're on the branch you want to merge into
git checkout ${branch}

# Merge to the next branch and type the commit message as described below.
git merge --no-ff --log ${branch_to_merge}
```

An optional helper script with detailed usage and help can be found here.

More info on this wiki page.

The tools are located in the CloudStack repository.

Commit messages

Merging a pull request

The commit message for a merging a pull request, should look like this:

> *Merge pull request #{pr number} from @ {user name of pr author}*

```
* pr/{pr number}:
  {list of commit messages}

Signed-off-by: {name and email of ACS committer that merges the pr}
```

It is important the pull request is *merged* to the branch. This script will help you do that (it will be added to the CloudStack repository).

Usage:

```
git pr 1234
```

Detailed usage and help can be found here.

Forward merging to the next branch

The commit message for forward merging, should look like this:

```
Merge fix release {source branch} to {current branch}

* {source branch}:
  {list of commit messages}
```

It is important the pull request is *forward merged* to the next branch. This script will help you do that (it will be added to the CloudStack repository).

Usage:

```
git fwd-merge some-branch
```

Detailed usage and help can be found here.

Important

We should all use the same tools to merge pull requests and do forward merges!

Gotcha: working with two origin repositories

When using these tools and this way of working on Apache CloudStack keep in mind that you work with two origins:

1. git-wip-us.apache.org/repos/asf/cloudstack.git (read/write)
2. and github.com/apache/cloudstack.git (read-only)

If you merge on a clone of one of them while it is behind on the other (or with its origin) you are merging on a commit that is no longer a HEAD. You will get an error when you push due to conflicts. At this point it is safest to throw away your merge, update (git fetch) you local clones and merge the PR again.

Initial merge of release branch to master

Originally written by Rajani Karuturi in PR 1071:

```
Initial merge of 4.6 to master
ignored pom.xml version number changes and changes to debian/changelog and engine/schema/src/com/cloud/upgrade
/DatabaseUpgradeChecker.java
Following commands were executed
1. git checkout 4.6
2. git pull --rebase
3. git checkout master
4. git pull --rebase
5. git fwd-merge 4.6
6. git diff --name-only | grep pom.xml | xargs git checkout --ours
7. git diff --name-only | grep pom.xml | xargs git add
8. git checkout --ours engine/schema/src/com/cloud/upgrade/DatabaseUpgradeChecker.java
9. git add engine/schema/src/com/cloud/upgrade/DatabaseUpgradeChecker.java
10. git checkout --ours debian/changelog
11. git add debian/changelog
12. # manually edited version number in tools/marvin/marvin/deployAndRun.py and tools/marvin/setup.py
13. git commit
14. git checkout -b "merge-46-to-master"
Send this as a PR
```

Monthly release schema

The monthly release schema looks like this:

Day 1: release of 4.x.0

Day 14: release of 4.x.1

Day 14: feature freeze 4.(x+1).0

Day 21: 4.(x+1).0 RC

Day 28: release of 4.x.2

Release management for 4.6

Based on these principles, Rajani Karuturi and Remi Bergsma volunteered to be the Release Managers of Apache CloudStack 4.6.

Release management for 4.7

Daan and Remi Bergsma

Release management for 4.9 LTS

Will Stevens

Release management for 4.10

Rajani Karuturi

Release management for 4.11 LTS

Rohit Yadav and Paul Angus

Thanks

Thanks to Rajani Karuturi, Daan, Miguel Ferreira, Wilder Rodrigues and all others for working on this with me (Remi Bergsma).