Goal

With branching and branch testability now being supported for Flutter & Dart releases, cherrypicking fixes will now be the preferred method to address issues for released software (beta and stable channels.) Stability of the release will be the overarching goal, so only high impactful and critical cherrypicks will be allowed across Dart and Flutter. This document outlines the process for requesting and approval of cherrypicks.

Note: This process applies to regressions from the previous release or serious bugs otherwise introduced by the current release. Feature work is not considered for cherrypicking and will need to wait for the next release.

The Cherrypick Reviewers

The Cherrypick reviewers will initially be composed of leads from product, engineering and program managers across Dart and Flutter. They will meet/discuss asynchronously on an as-needed basis assuming there are requests to review or prioritize. Longer term, we may include contributors as well. The individual (Dart, Flutter) groups will meet to review cherrypick requests from their repos as needed, organized by their TPMs. The full group will meet when cherrypicks are approved by the individual groups and consideration for a hotfix release is required.

As we get into the stable release period, the unified group will still need to come to consensus for approval of **all** cherrypicks, even if they are for a certain area, as risk will need to be assessed across Dart and Flutter.

Process Flow

1. Issue/bug is filed
    1. "cp: review" can be added here by filer or community
2. Issue enters triage process
    1. Top level triage
        1. Applies appropriate team label
        2. Can apply "cp: review" label if it appears to have appropriate severity or impact.
    2. Engineering triage
        1. Can apply "cp: review" label if it appears to have appropriate severity or impact.
        2. If Cherrypick request label is on the issue, agree on severity/impact and prioritization with the Cherrypick review
3. **Issue is fixed and checked into master.**
    1. Pre and post-submit tests executed as normal for verification.
    2. Cherrypick reviewers may influence prioritization depending on the severity of the issue during the bug fixing process.
4. Fixed issue with "cp: review" is reviewed by the individual Cherrypick reviewer(s).
    1. TPgM/PM/Eng involved to make a decision based on factors that are different for beta and stable and the age of the branch.
        1. **Impact criteria:**
            1. Early beta stabilization (first 2 weeks of beta): High impact P1/P0 affecting multiple customers / would be a P0 in stable / regressions / security issues
            2. Late beta stabilization: P0 issues / P1 issues affecting larger customers / regressions / security issues
            3. Stable: P0 issue, affecting large base of customers, high % of user apps | importance factor / security issues

            1. crashing on x% of phone ecosystem
            2. breaking developer flow with no easy workaround
        2. **Risk criteria:**
            1. Is it a manual or automated merge?
            2. How many lines of code, what areas of the stack is it touching?
            3. Testability of the cherrypick.
        3. **Release timing**
            1. How close is the next beta/stable release?
5. If the cherrypick is deemed necessary and approved by the individual team reviewers, the full set of reviewers (Flutter+Dart) will then review using the same factors as above, with attention to full stack risk.
6. If denied, the issue is updated with denial reason and ideally a target release.
7. If approved
    1. TPM/Developer will create PR into the vX.Y.Z branch
        1. for Dart, this will be the beta branch until the vX.Y.Z branch support or similar is built.
        2. If pre-submit tests fail, a risk assessment will need to be performed with input from the cherrypick reviewers.
    2. In case of a manual merge, see the below process on Resolving Merge Conflicts.
    3. Post-submit tests are triggered on branch
        1. In case of multiple cherrypicks/issues this will be done after a batch
8. When tests pass
    1. when there is a lull in CPs / or a priority CP has been merged, a new release will be made available through the release process.

Resolving Merge Conflicts

In the case that an approved PR requires a manual merge to the desired release branch, the release manager (the TPM or developer who is conducting the release) will have the PR's original author create a new change to resolve the conflict.

1. The release manager identifies a PR that will not merge cleanly with the release branch.
2. The release manager will notify the PR author on GitHub on the release PR with a link to this document and a request to create a new PR merging the change into the release branch. This message can also be replicated in an e-mail and in chat to ensure the author is reached. If the author cannot be reached, one of the PR's approvers will be selected to do the merge.
3. The PR author will fetch upstream, and then checkout the release branch that the release is targeting. For example:

```
$ git fetch upstream && git checkout -b $RELEASE_BRANCH upstream/$RELEASE_BRANCH
```

4. The PR author should then cherrypick the commit from `master` that maps to their approved PR:

```
$ git cherry-pick $CHERRYPICK_COMMIT
```

5. It is expected that the previous command will have a merge conflict. The PR author will resolve this conflict locally in their editor. They will run relevant tests and analysis to ensure the merged code is correct.
6. The PR author will create a new PR. It is recommended to name the branch you push to your fork the same as the upstream release branch, to skip an additional step. In the new PR UI, there is a drop-down menu for

the base branch the PR will merge into: this should be set the same `RELEASE_BRANCH` that the release is based on. The PR author will add the release manager as a reviewer.

7. If the PR author did not name the branch on their fork the same as the upstream release branch, the Flutter GitHub bot will reset the merge target branch of PR to `master`. If this happens, the PR author will press the `edit` button in the top right of the PR UI, which will reveal a drop down menu to change the `base` branch back to the release branch. The bot will not reset the PR a second time.
8. The release manager will review the PR and validate pre-submit CI builds. After approval, they will merge the PR to the release branch.
9. The release manager will rebase their initial release PR against the release branch with the new cherrypick commit. If there are any conflicts rebasing, the commit with the conflict will undergo this same process.