[Editor note: Originally by Misty Linville at Google, mistyhacks@google.com]

Objective

This is a proposal to modify the branch management strategy for Kubernetes docs to meet the following objectives:

- Make it easier to understand what branch to make your PR against for a given unit of work
- Allow for continuous improvement for the currently-published docs
- Have “release trains” for documenting upcoming new branches

Current situation

- New feature work goes into a feature branch like `release-1.12`, which is regularly rebased on `master` (we have also tried regularly merging `master` into the feature branch)
    - Rebasing means that every non-merged PR against the feature branch needs to be rebased manually each time we rebase the feature branch.
    - This is done by the PR author or (if PR settings allow) by the release manager.
    - This could introduce unaudited changes into the PR, and can also surprise the PR owner who may not remember to rebase their local work on their fork.
- Continuous improvement goes directly into `master`
- Archive branches are maintained for old releases, but very rarely updated
- Developers are surprised by the feature branch strategy and expect all new work to go into `master`. This means we have to ask them to rebase their PR on the feature branch often.

Proposal

- Continuous improvement work goes into `master`.
- Not-yet-released feature work still goes into a feature branch like `dev-1.12`.
    - In theory there could be more than one upcoming release in development at once, though it doesn’t happen often.
    - When a `dev-` branch is merged into master, that dev- branch can be removed as we never need to do any more pre-release work in that branch.
- Right after a release happens, and until the next release happens, `master` represents the docs for the current release, as well as continuous improvement.

- When the release is no longer current, a new branch is created as the archive for that release (named something like `release-1.12`) Any post-release version-specific docs updates happen there (this is rare).
- Big multi-person efforts like content reorg, information architecture, or big refactors go into feature branches based off `master`, and merged into `master` when ready.

Mechanics

1. Release manager creates and manages a given `dev-` branch. This is a rotating duty.
    a. Create the `dev-` branch.
    b. Merging PRs that are ready
    c. Regular (weekly?) merge `master` into the dev- branch to get continuous improvements by making a PR.
    d. Releasing means:
        i. Create and push the final tag for the prior release by tagging master just before merging the dev- branch into master
        ii. Based on that tag, create and push the branch that represents the archive for the prior release
            1. For instance, the RM for 1.13 would create and push the `release-1.12` branch based on the final 1.12 tag
        iii. Create a PR to merge the `dev-` branch into `master`
        iv. Create the initial tag for the new release right after merging the `dev-` branch into `master`
        v. Delete the `dev-` branch since it's no longer needed, maybe.
            1. Some people say you should never delete branches so in that case we just wouldn't accept new PRs against it.
2. Feature branch manager is effectively a release manager for a feature branch, without all the tagging and extra branches.
    a. TLDR: To "release" a feature, you only need to merge its branch into `master`.

Why this is better than what we have now

- The `dev-` naming scheme makes it a little more obvious that those are for pre-release content
- Merging workflows work better in Github
- Doing more with PRs in the Github UI is more user-friendly for people who are not Git CLI experts

- Gets rid of the busy work of keeping `release-XYZ` up to date with master during the time that they are the same

Internationalization workflows

https://docs.google.com/presentation/d/14x_kLkk6cAvF04uDwIYTGWuE_bFxPY83MvrCb9nHfxE/edit#slide=id.g4152337e7a_2_188

Meeting Notes

9/10/2018
Publishing bot & staging repos

- https://github.com/kubernetes/publishing-bot
- https://github.com/kubernetes/apiserver
- Munges the above two repos: https://github.com/kubernetes/kubernetes/tree/master/staging/src/k8s.io/apiserver