

RELEASE FLOW PROPOSAL

PETRO KOLOSOV

ABSTRACT. In this document software product release process is proposed and discussed.

CONTENTS

1. Introduction	1
1.1. Release process	2
1.2. Hotfix strategy	4
2. Conclusions	4
References	5

1. INTRODUCTION

Release flow is a set of steps to perform to release upcoming version of software product. Main aim of this document is to present simple and working model of software release using semantic versioning [1], azure pipelines, and mainline development [2]. Mainline development is also known as GitHub flow. Current document is motivated by Microsoft's "Adopt a Git branching strategy", see [3]. Below picture shows main idea of GitHub flow itself

Date: January 3, 2025.

Key words and phrases. Software engineering, DevOps, Software release, GitHub flow, GitLab flow, Azure DevOps, Azure pipelines, Semantic versioning, GitVersion, CI/CD .

Sources: <https://github.com/kolosovpetro/ReleaseFlowProposal>

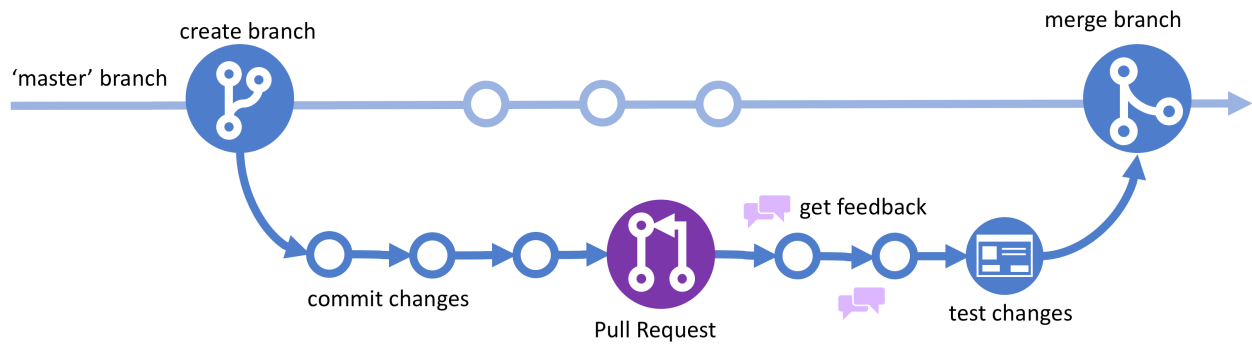


Figure 1. GitHub Flow diagram.

- **Master** – branch that contains tested, validated and verified code, ready to be released.
- **Feature** – branch that contains implementation of new feature according to sprint plan. Feature is branched from the **master** HEAD.
- **Bugfix** – branch that contains non-critical bug fix. Bugfix is branched from the **master** HEAD.
- **Release/v*** – branch that contains upcoming release state of software product. Release engineer decides what commit to be base of release branch. **Release/v*** is branched from the **master**. **Release/v*** is considered to be long-living branch, it should not be deleted after code is deployed. **Release/v*** branch is used to support current release of software product, after new release occurs team decides whereas to keep or to delete old release branch. After release complete **master** branch cherry-picks [4] from it.
- **Hotfix** – branch that contains critical bug fix. It is used to patch production environment and must be released as quick as possible.

1.1. **Release process.** Having all above, assume we have initial semantic version on our **master** branch as `v0.1.1`, and we must release upcoming version, our steps to perform are:

- (1) Software engineer creates pull request from recent **feature** branch to **master** branch, this pull request triggers Continuous Integration (CI) to start, CI runs tests, code quality checks etc., but deployment won't be started yet, only CI.
- (2) After all CI checks passed, pull request reviewed by team and every comment from code review is fixed – it is ready to be merged from **feature** branch to **master** branch. No CI/CD pipeline triggered by the merge.
- (3) Next, release engineer reviews software product changes using **CHANGELOG** or **git compare**. Release engineer makes decision which part of semantic version to increment. For example, release engineer decided that minor version should be incremented then our version becomes, for example **v0.1.1 -> v0.2.0**
- (4) Release engineer creates new release as follows
 - Checkout to release branch: `git checkout -b release/v0.2.0`
 - Work on release committing some minor fixes
 - Push release branch to remote: `git push origin release/v0.2.0`
 - Create tag: `git tag -a v0.2.0 -m "Release v0.2.0"`
 - Push tag: `git push origin v0.2.0`
- (5) After new tag is pushed the CI/CD pipeline is triggered by that event [5]. There are three deployments scheduled: DEV, QA, UAT. Environments QA and UAT are to be approved by designated personnel before deployment starts, DEV to be deployed without manual approve.
- (6) Finally, **master** branch cherry-picks [4] from **release/v0.2.0** after deployment is complete.

Entire release process is shown on the picture below

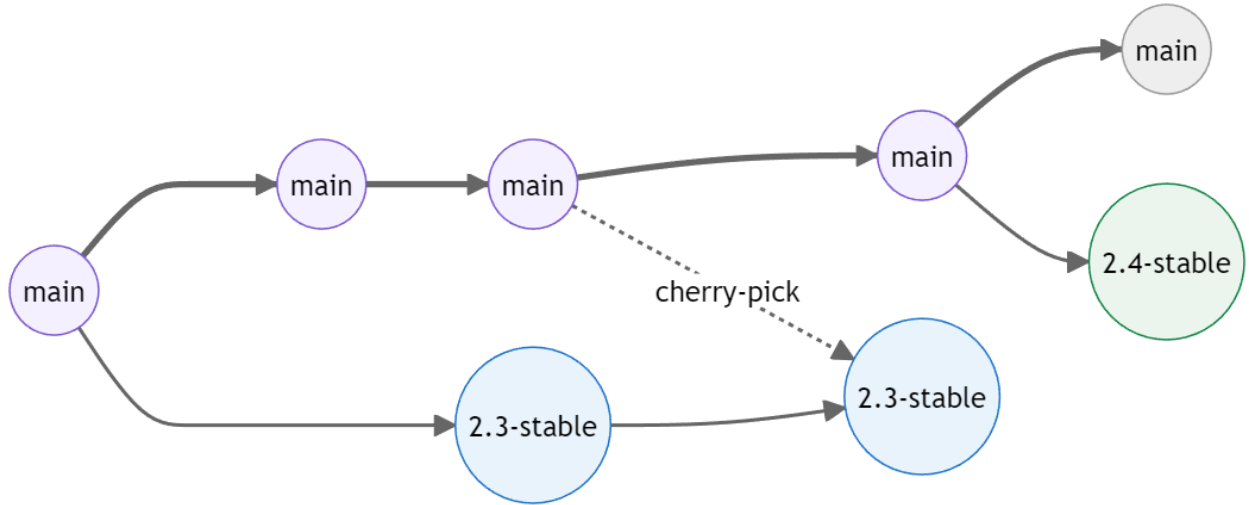


Figure 2. GitLab Flow diagram [6].

1.2. **Hotfix strategy.** Assume that our current released version of software product is `v0.2.0` and there is a critical bug appears. In order to release a hotfix the following set of steps to be executed:

- (1) Hotfix to be assigned to a software engineer.
- (2) Software engineer fixes critical bug and creates a pull request: `hotfix/id -> master`.
Yes, pull request is done to the `master` branch.
- (3) Pull request `hotfix/id -> master` is reviewed by team and merged.
- (4) Release engineer cherry-picks [4] recently merged hotfix from the `master` branch to the `release/v0.2.0` branch. Note that branch `release/v0.2.0` is long-living and kept minimum until next release.
- (5) Release engineer increments patch part of semantic version, e.g `v0.2.0 -> v0.2.1`.
- (6) Release engineer creates and pushes new tag `v0.2.1`.
- (7) Hotfix deployment process is started after new tag is pushed.

2. CONCLUSIONS

In this document software product release process is proposed and discussed. Few useful GIT commands worth to remember:

- `git tag -a v0.1.0 -m "my version 0.1.0"`
- `git tag -d <tag_name>`
- `git push origin <tag_name>`
- `git push --delete origin <tag_name>`

REFERENCES

- [1] Semantic Versioning Docs. Semantic Versioning 2.0.0, 2023. <https://semver.org/>.
- [2] GitVersion Docs. Mainline Development, 2023. <https://gitversion.net/docs/reference/modes/mainline>.
- [3] Microsoft Documentation. Adopt a Git branching strategy, 2022. <https://learn.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops>.
- [4] Atlassian Docs. Git Cherry Pick, 2023. <https://www.atlassian.com/git/tutorials/cherry-pick>.
- [5] Microsoft Documentation. Build Azure Repos Git or TFS Git repositories, 2023. <https://learn.microsoft.com/en-us/azure/devops/pipelines/repos/azure-repos-git>.
- [6] GitLab Docs. Introduction to GitLab Flow, 2023. https://docs.gitlab.com/ee/topics/gitlab_flow.html.

Version: Local-0.1.0

SOFTWARE DEVELOPER, DEVOPS ENGINEER

Email address: kolosovp94@gmail.com

URL: <https://kolosovpetro.github.io>