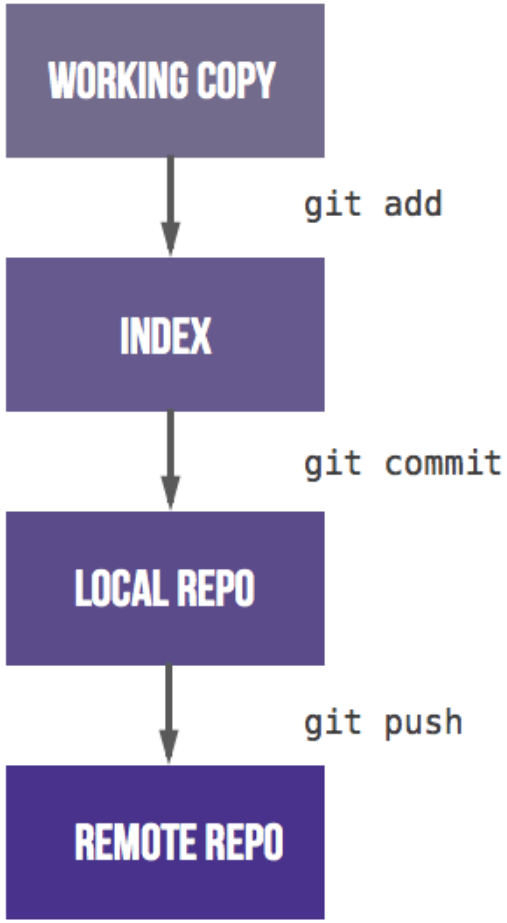
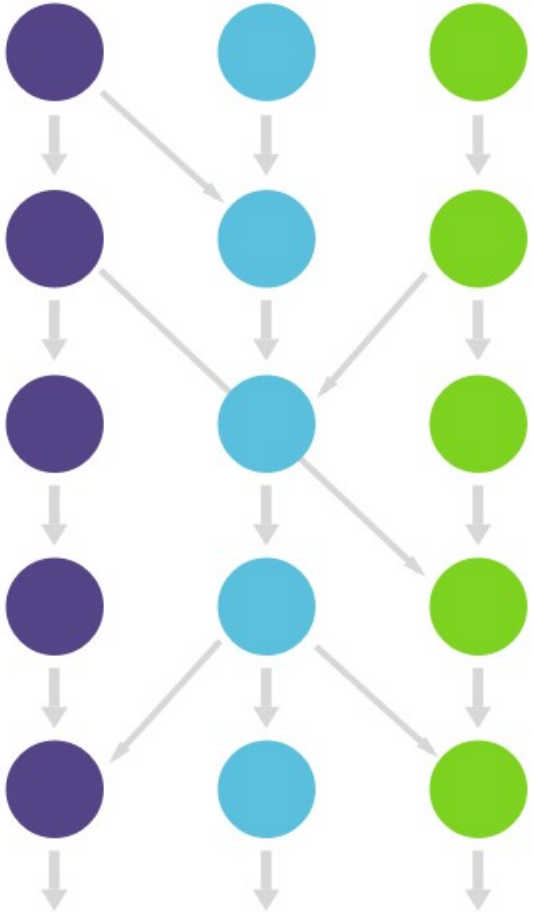
GIT, GIT FLOW, CI/CD

GIT

Переход от централизованного контроля версий к распределенному

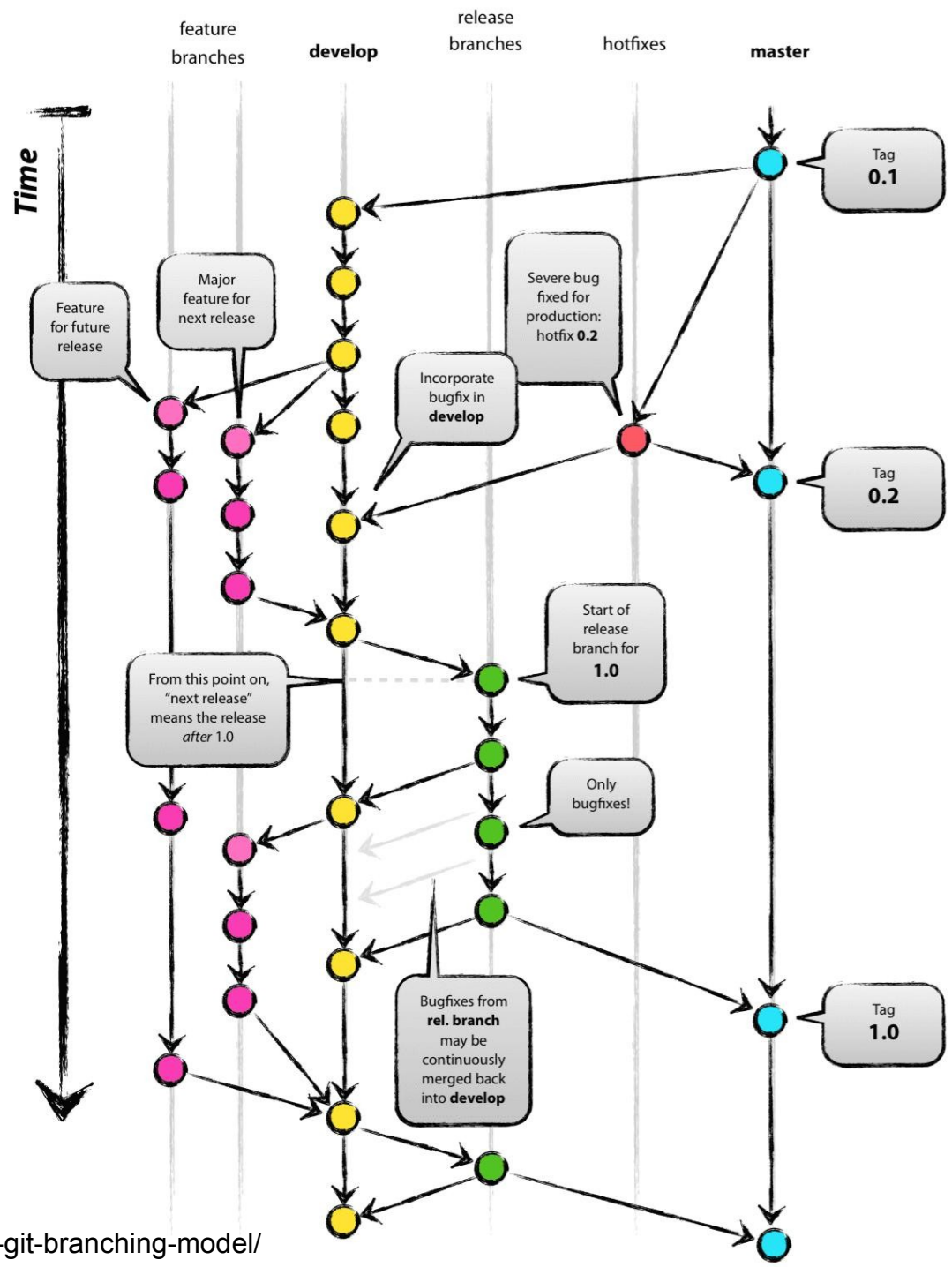


Упрощенное создание и слияние веток



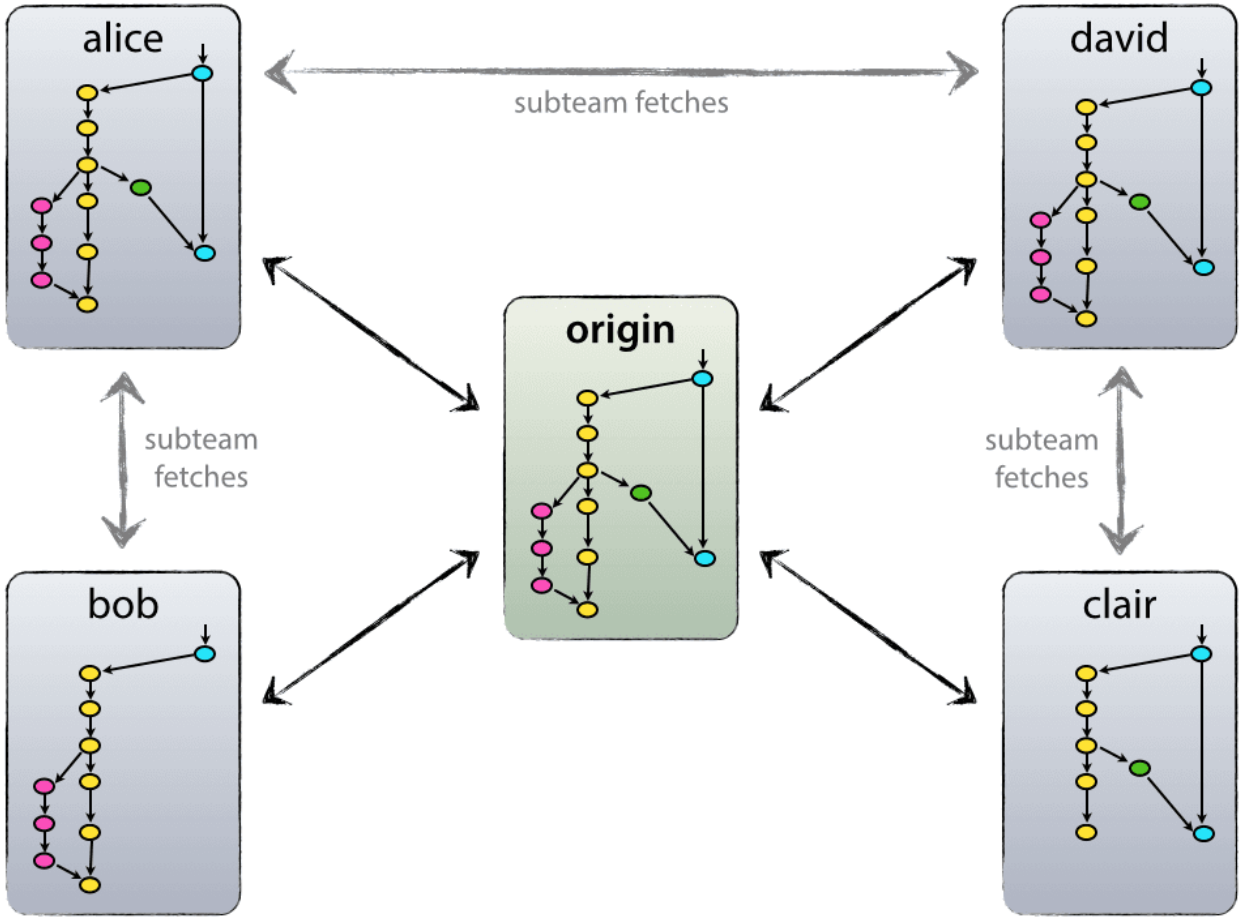
https://docs.gitlab.com/ee/topics/gitlab_flow.html

GIT FLOW



https://nvie.com/posts/a-successful-git-branching-model/

Decentralized but centralized



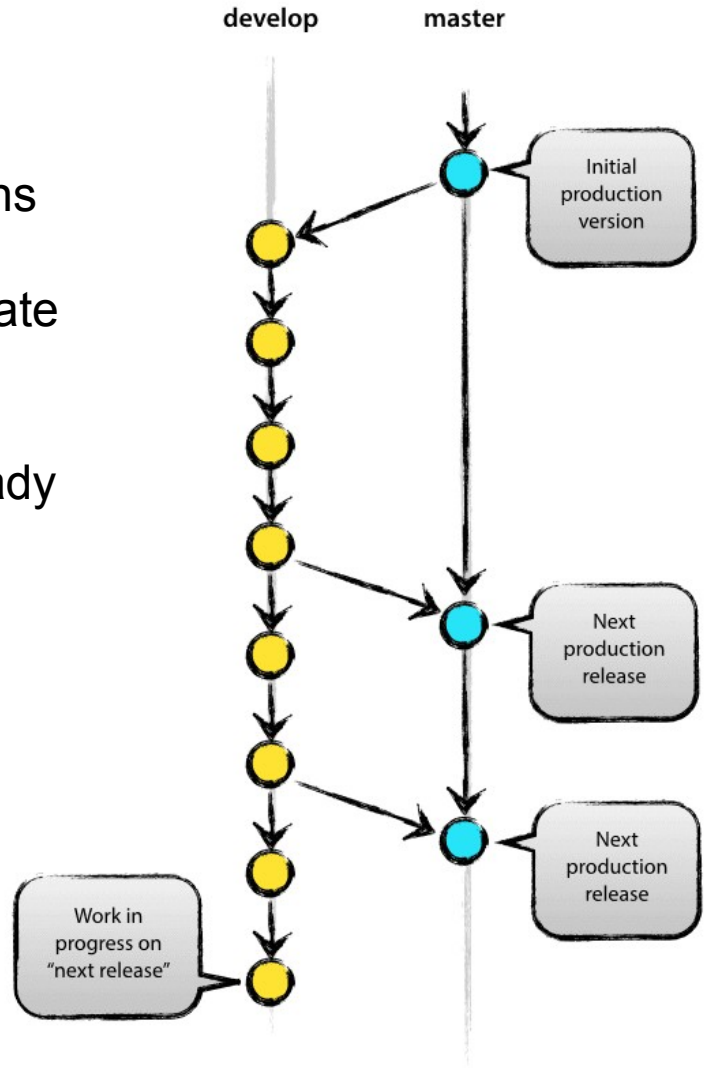
https://nvie.com/posts/a-successful-git-branching-model/

Main branches

Master is a production ready code
Probably tagged with release versions

Develop is a current development state

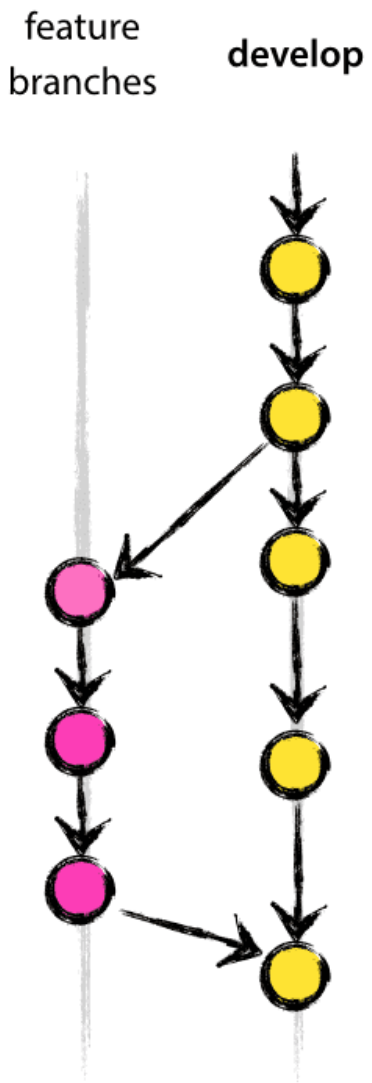
Commit only to Develop
Merge to master when release is ready



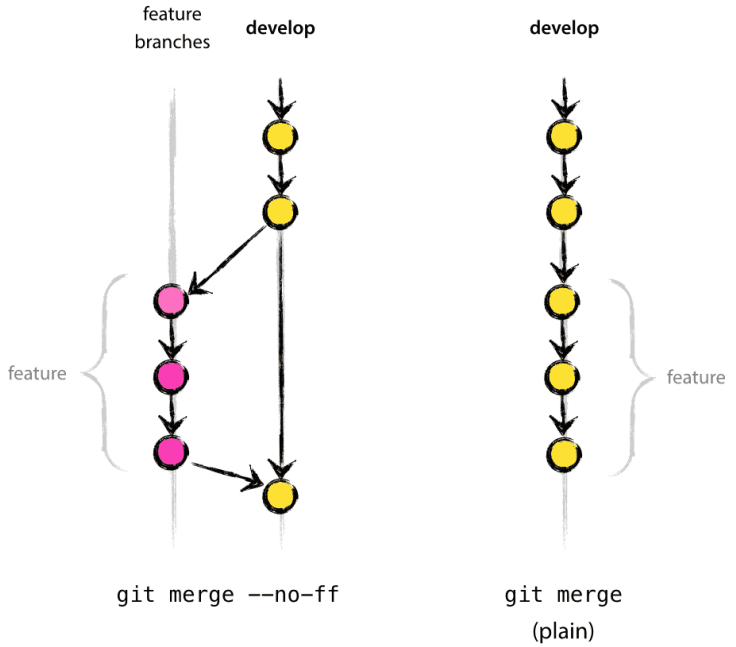
https://nvie.com/posts/a-successful-git-branching-model/

Support branches: features

May branch off from:
develop
Must merge back into:
develop
Branch naming convention:
anything except master, develop, release-*, or hotfix-*
Feature branches typically exist in developer repos only, not in origin.




```
Create a myfeature branch from develop:
$ git checkout -b myfeature develop
Switched to a new branch "myfeature"

Incorporate finished feature to develop:
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff myfeature
Updating ea1b82a..05e9557
(Summary of changes)
$ git branch -d myfeature
Deleted branch myfeature (was 05e9557).
$ git push origi
```



https://nvie.com/posts/a-successful-git-branching-model/

Support branches: releases

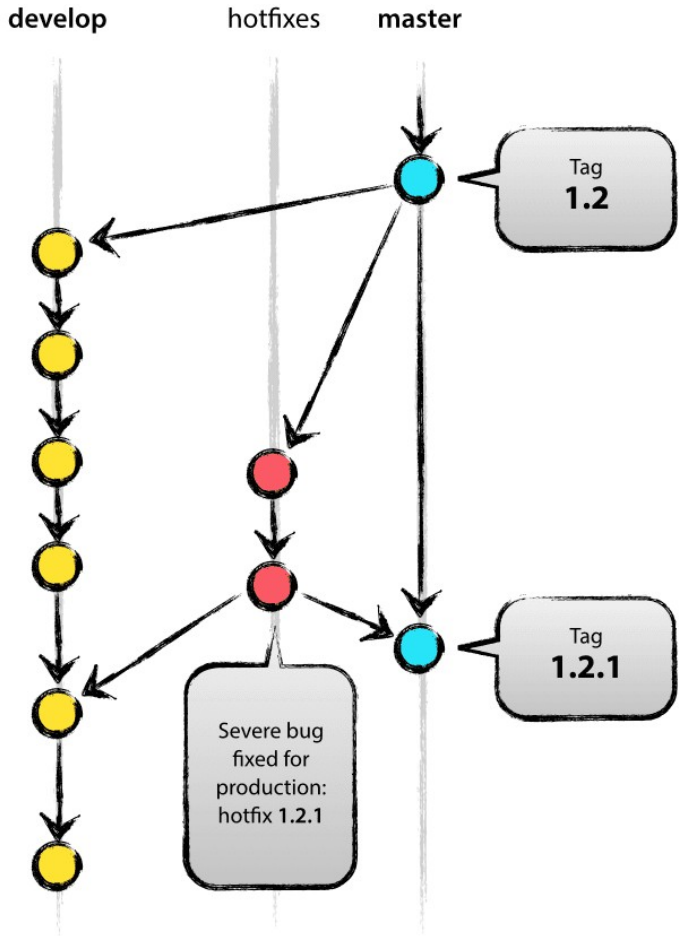
May branch off from:
develop
Must merge back into:
develop and master
Branch naming convention:
release-*

```
Creating a release branch:
$ git checkout -b release-1.2 develop
Switched to a new branch "release-1.2"
$ ./bump-version.sh 1.2
Files modified successfully, version bumped to 1.2.
$ git commit -a -m "Bumped version number to 1.2"
[release-1.2 74d9424] Bumped version number to 1.2
1 files changed, 1 insertions(+), 1 deletions(-)

Finishing a release branch:
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
$ git tag -a 1.2
Push tag to origin remote
$ git push origin 1.2

Merge changes to develop:
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)

Delete branch:
$ git branch -d release-1.2
Deleted branch release-1.2 (was ff452fe).
```

https://nvie.com/posts/a-successful-git-branching-model/

Support branches: hot fixes

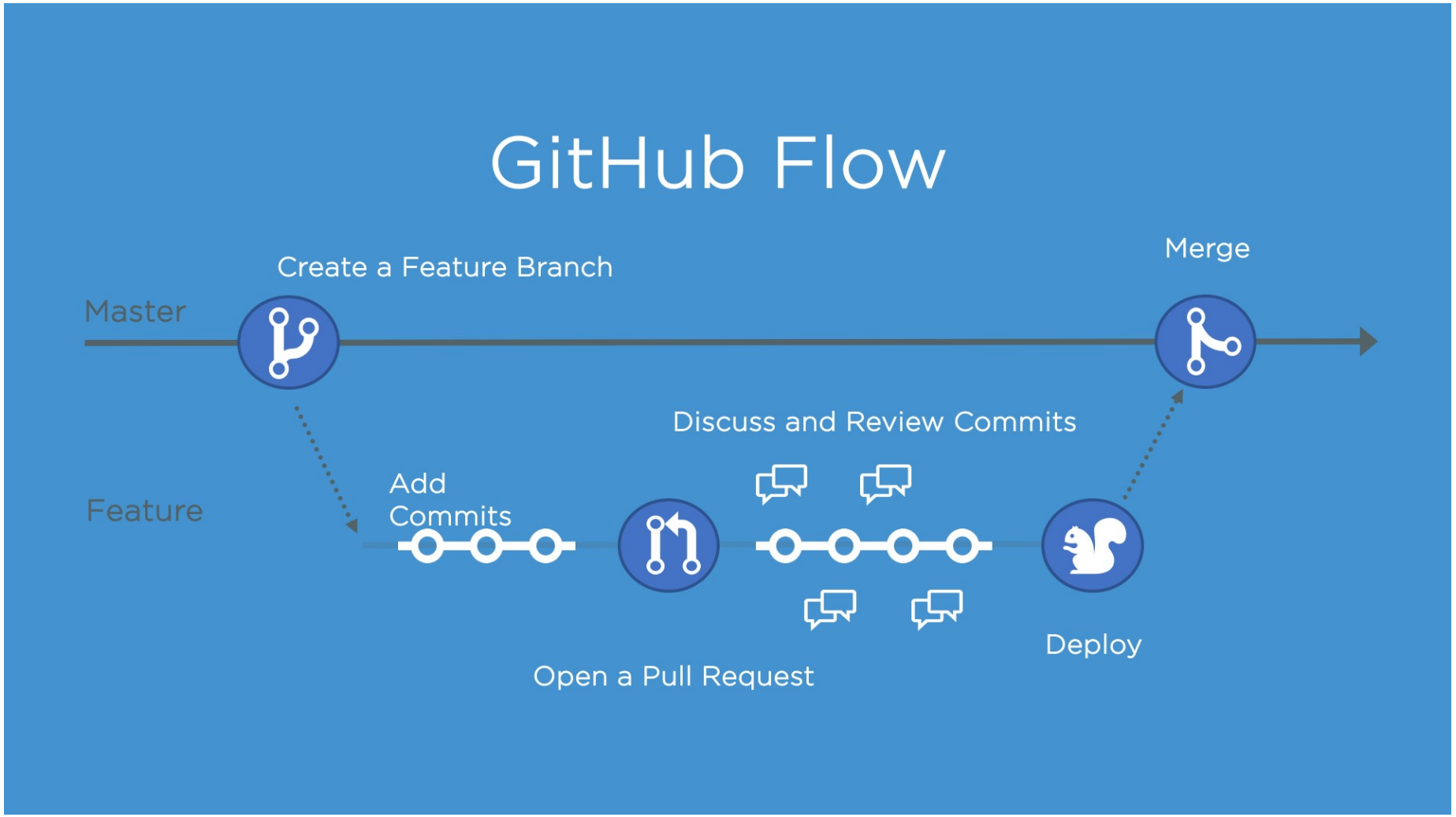
May branch off from:
    master
Must merge back into:
    develop and master
Branch naming convention:
    hotfix-*




```
Creating a hotfix branch:
$ git checkout -b hotfix-1.2.1 master
Switched to a new branch "hotfix-1.2.1"
$ ./bump-version.sh 1.2.1
Files modified successfully, version bumped to 1.2.1.
$ git commit -a -m "Bumped version number to 1.2.1"
[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1
1 files changed, 1 insertions(+), 1 deletions(-)
$ git commit -m "Fixed severe production problem"
[hotfix-1.2.1 abbe5d6] Fixed severe production problem
5 files changed, 32 insertions(+), 17 deletions(-)

Finishing a hotfix branch:
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)
$ git tag -a 1.2.1
Push tag to origin remote
$ git push origin 1.2.1

Include fix to develop and delete hotfix branch:
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)
$ git branch -d hotfix-1.2.1
Deleted branch hotfix-1.2.1 (was abbe5d6).
```

https://nvie.com/posts/a-successful-git-branching-model/

Alternatives: GitHub Flow



https://github.com/dorren/release-based-workflow/issues/1

Alternatives: CI/CD

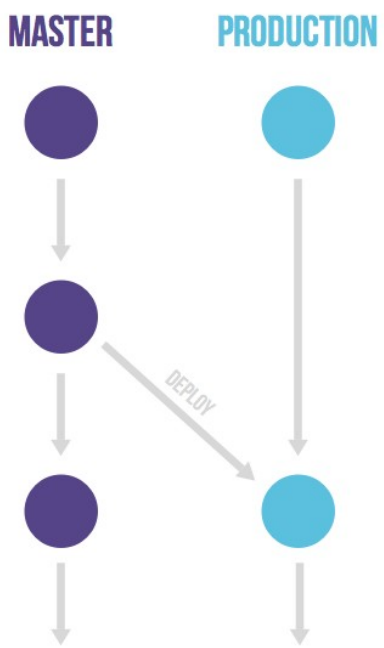
Continuous Integration & Continuous Delivery:
- Have one always working code version
- No Feature Branches (!)
- Commit often, integrate to master (main) daily
- Feature flags to hide incomplete features
- Use Green/Blue (Red/Black) deployment
- Canary deployment (backend)
- Dark releases (frontend)

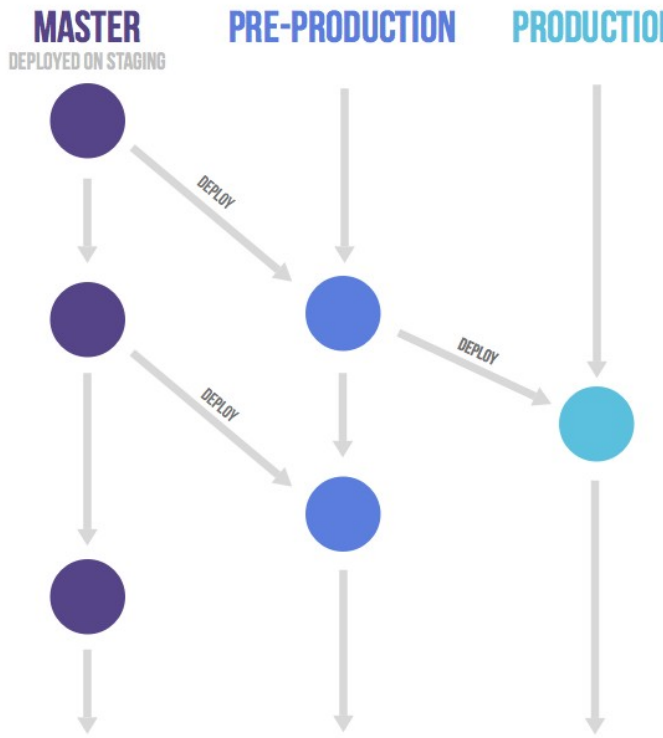
Profit: faster feature development
and more happy life for developers!

See:
https://youtu.be/v4Ijkq6Myfc
https://habr.com/ru/company/flant/blog/471620/

https://github.com/dorren/release-based-workflow/issues/1

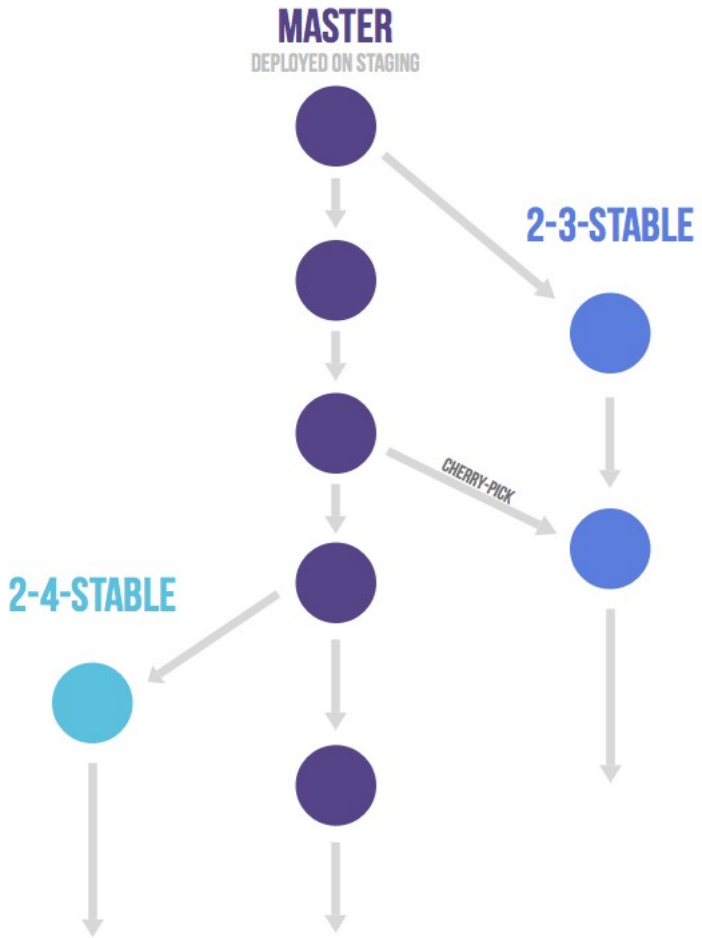
Alternatives: Production & Environment branches with GitLab Flow





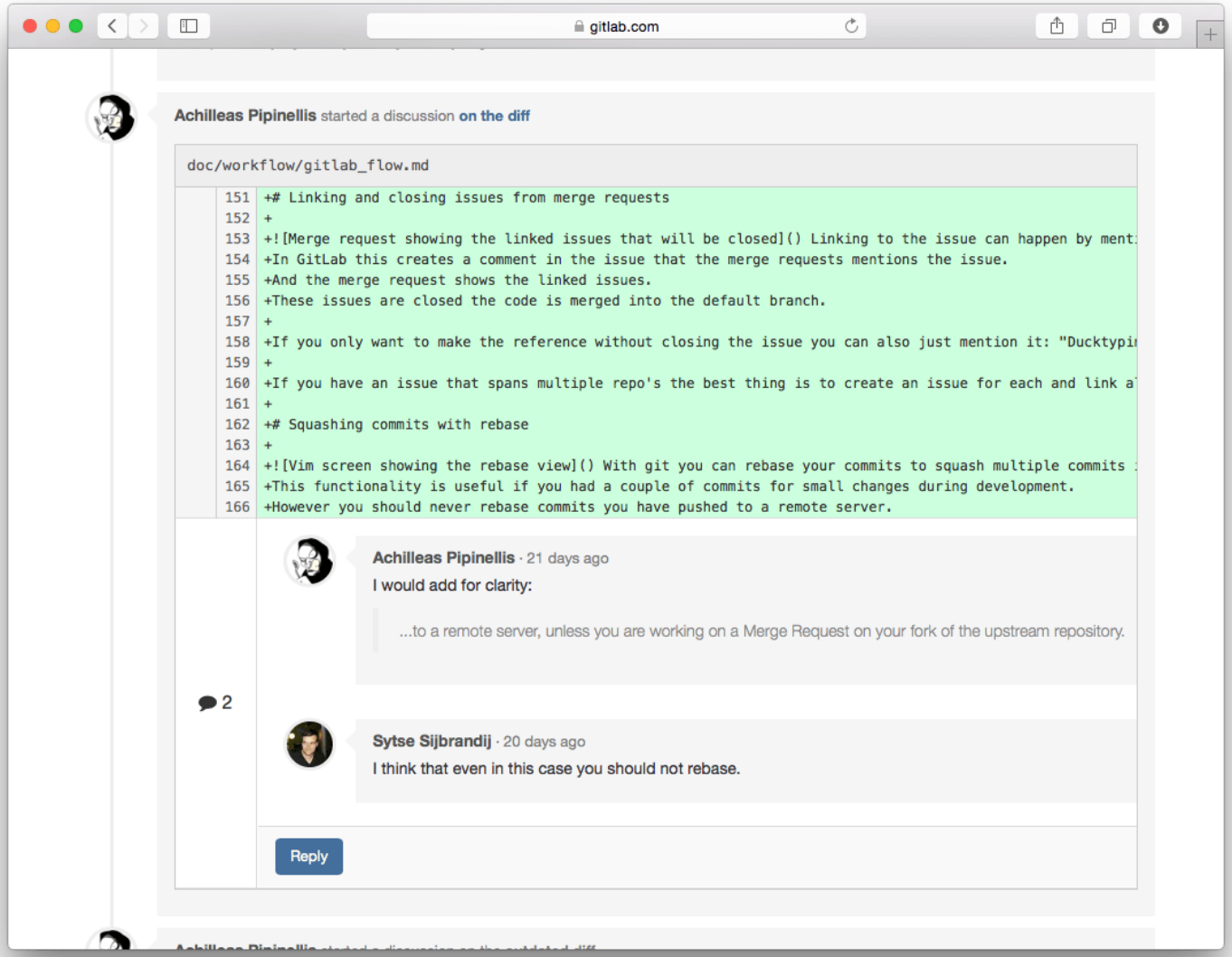
https://docs.gitlab.com/ee/topics/gitlab_flow.html

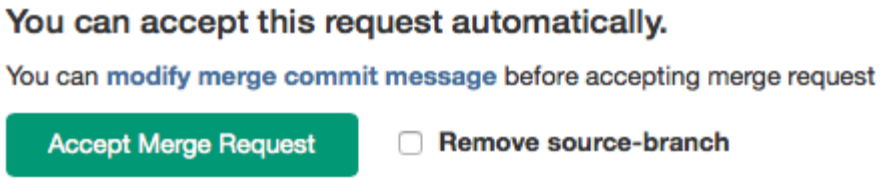
Alternatives: Release branches with GitLab Flow



https://docs.gitlab.com/ee/topics/gitlab_flow.html

Merge/Pull requests






```
Manually remove source branch:
git push -d origin <branch_name>
```

https://docs.gitlab.com/ee/topics/gitlab_flow.html#mergepull-requests-with-gitlab-flow

Git pull

Чтобы забрать изменения из удаленной ветки *dev* репозитория *origin* и слить их с изменениями в текущей ветке, где мы находимся, выполним:

```
1.  git pull origin dev
```

Это аналогично двум последовательным командам:

```
1.  git fetch origin
2.  git merge origin/dev
```

Обратите внимание, что перед выполнением команды *pull* все текущие изменения должны быть закоммичены, так же, как перед выполнением команды *merge*. Иначе слияние не выполнится.

https://sysout.ru/komanda-git-pull/

Практика

- Добавиться в проект GitLab
- Клонировать репо проекта
- Сделать ветку фичи от мастер (main)
- Дописать строку в README.md
- Закоммитить и запушить фичу в свою ветку
- Сделать мерж реквест и попросить коллегу смержить в мастер (main) свою ветку фичи
- Выполнить мерж реквест по просьбе коллеги, предварительно сделать пулл ветки и посмотреть
- Сделать ветку релиза от мастер (main) и поменять в ней версию в файле README.md
- Влить ветку релиза в мастер и пометить тегом версии релиза, устранить конфликт если есть
- Запушить изменения в мастер и в ветке релиза на origin и посмотреть в GitLab

Ссылки

https://docs.gitlab.com/ee/topics/gitlab_flow.html
https://github.com/dorren/release-based-workflow/issues/1
https://nvie.com/posts/a-successful-git-branching-model/
https://youtu.be/v4ljkq6Myfc
https://habr.com/ru/company/flant/blog/471620/