HACK BASICS

Git theory & practice

F0RTHSP4CE

Version Control System (VCS)

- FTP
- Apache Subversion

- Git

Why choose Git?

- free & opensource
- fast (operates locally)
- no need for external backups
- simple branching
- decentralised

Let’s start practice

- Github
- Atlassian Bitbucket
- Gitlab

https://gitlab.f0rth.space

register here

Create project and clone

- `git init` - create repository locally

  `Create Internal repo in gitlab`

- `git clone` - copy repo to local machine

  `git clone https://gitlab.f0rth.space/user/your-repo.git`

- `git config --list` - check current config
- `vi ~/.gitconfig` - edit user config

Let’s make first changes

`make change in README.md locally`

- **git status** – view changed files

1) untracked
2) modified
3) ignored (!filename)

- **git add** – add files for commit

```
git add ./*
git status
```

- **your-repo/.gitignore** – ignored files

```
echo “*.ini” > .gitignore; touch ex.ini
git add ./*
```

First commit

- **git commit** – record changes to the repository

```
git commit -am “change README.md”
```

! This commit will ___<your commit message>_____ !

- **git commit --amend** – replace last commit
- **git log** – show commit history
- **HEAD pointer** – points on last commit (local and remote)
  HEAD~2 – 2nd commit before last

Work with remote repository

- `git push <remote> <branch>` - upload changes

  ```
  git push origin main
  ```

- `git remote get(/set)-url <remote>`
  `git remote add/remove <remote>`
  `git remote -v` - show list of remote repos
- **`git fetch –`** download changes from remote
- **`git pull –`** download changes from remote and merge

Cancel commits

- `git revert <commit>` – adds revert commit to log

  ```
  git revert --no-edit HEAD~1
  ```

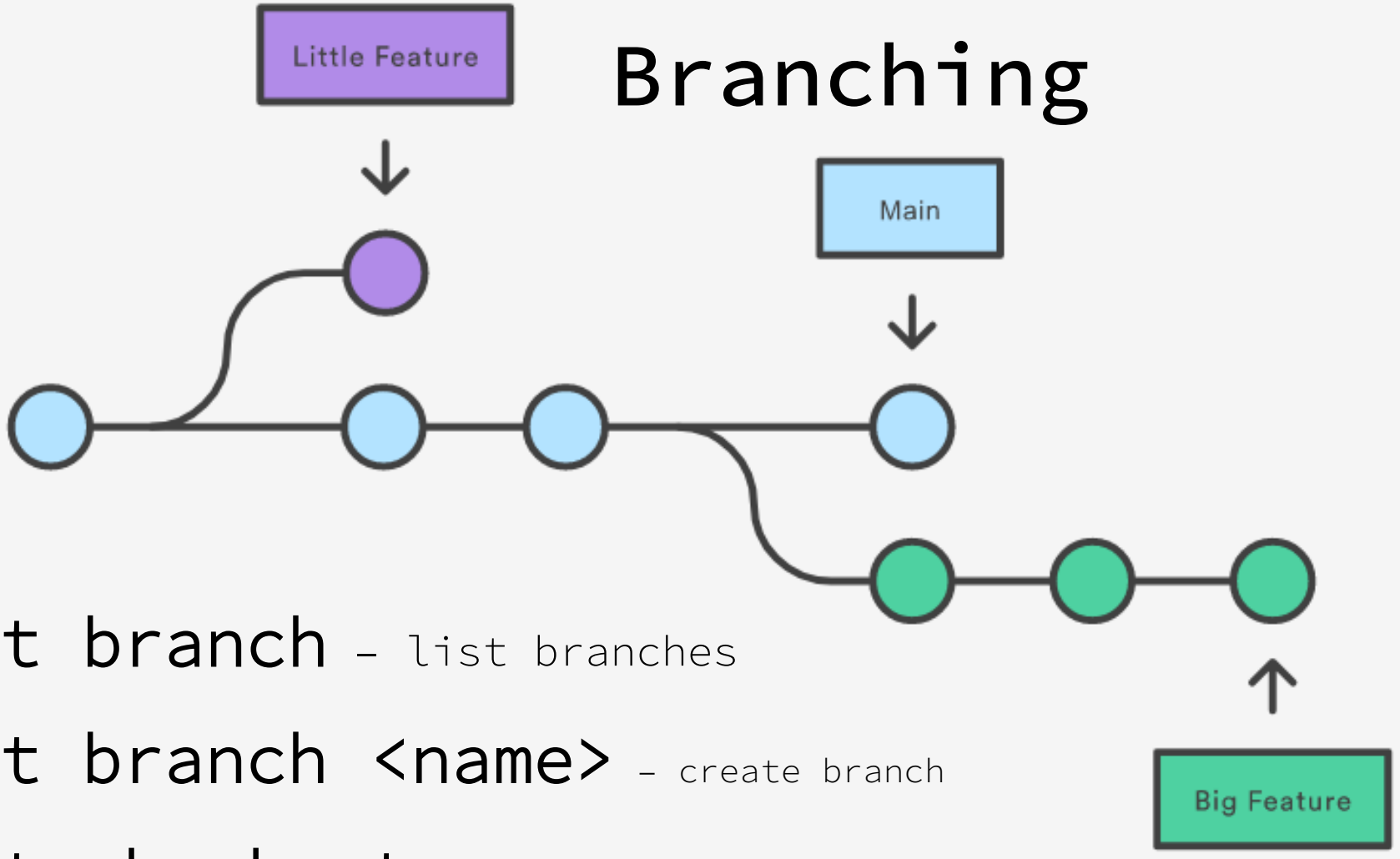
- `git reset --soft <commit>` - moves HEAD, leaves changes

  ```
  git reset --soft HEAD~1
  ```

- `git reset --hard <commit>` - delete from remote
  `git push --force`

  ```
  git push
  git reset -hard HEAD~1; git push -f
  ```

Branching



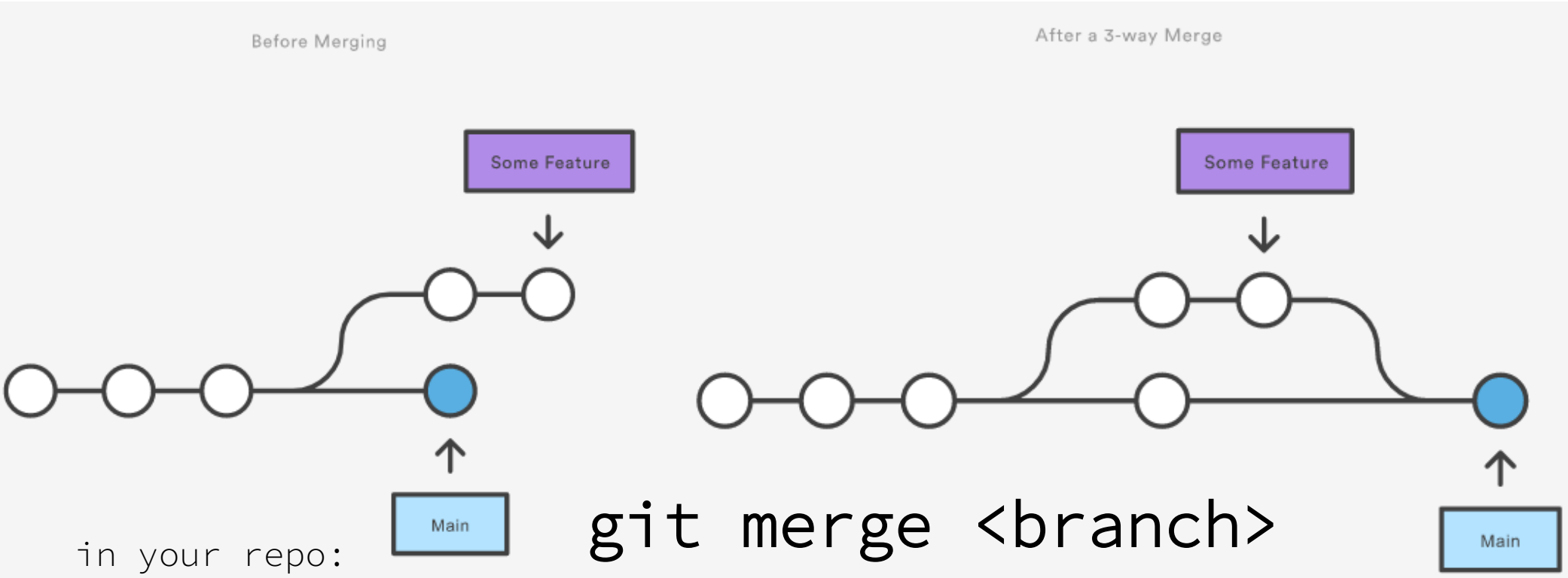
- `git branch` - list branches
- `git branch <name>` - create branch
- `git checkout` - switch to branch

practice

- in your repo, create file color.yml in main branch, write there “red”, commit and push to remote

- clone your left neighbors repo
- create branch “fix” and checkout it
- create color.yml in this branch and write there “green”
- commit it and push to remote
- create branch “feature” and checkout it
- create file feature.py
- commit and push to remote

Merge



git merge <branch>

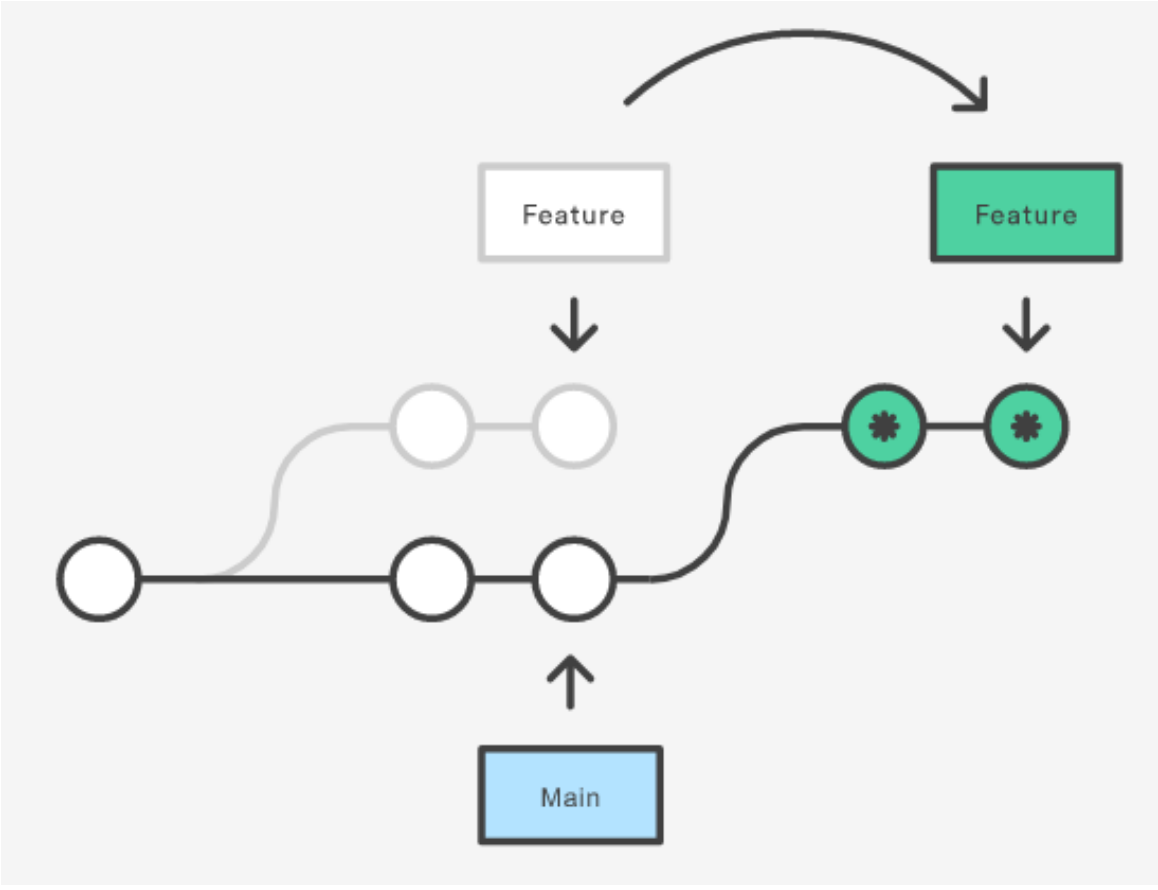
in your repo:

```
git merge fix
vi color.yml (resolve conflict)
git commit
```

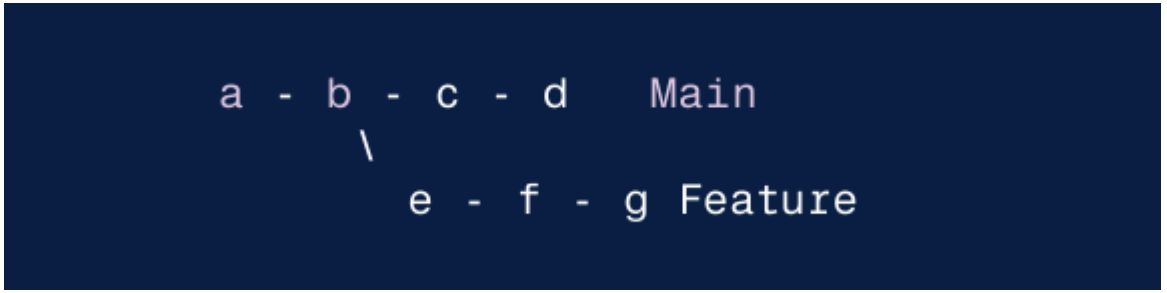
Rebase

- `git rebase <base>`
- `git rebase -i <base>`

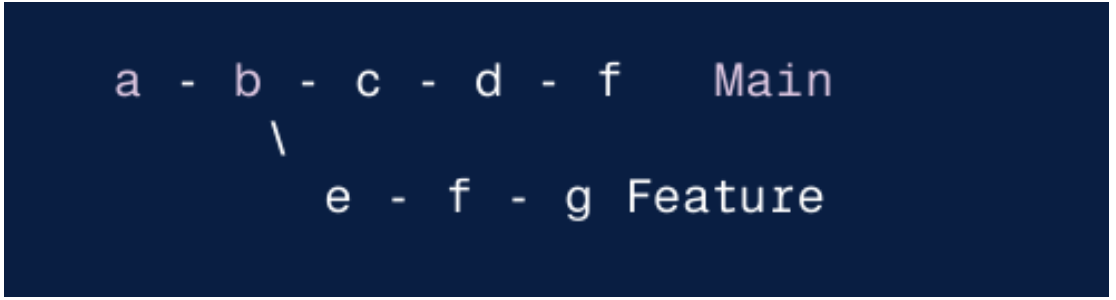
```
git checkout main
git rebase -i feature
```



Cherry-pick

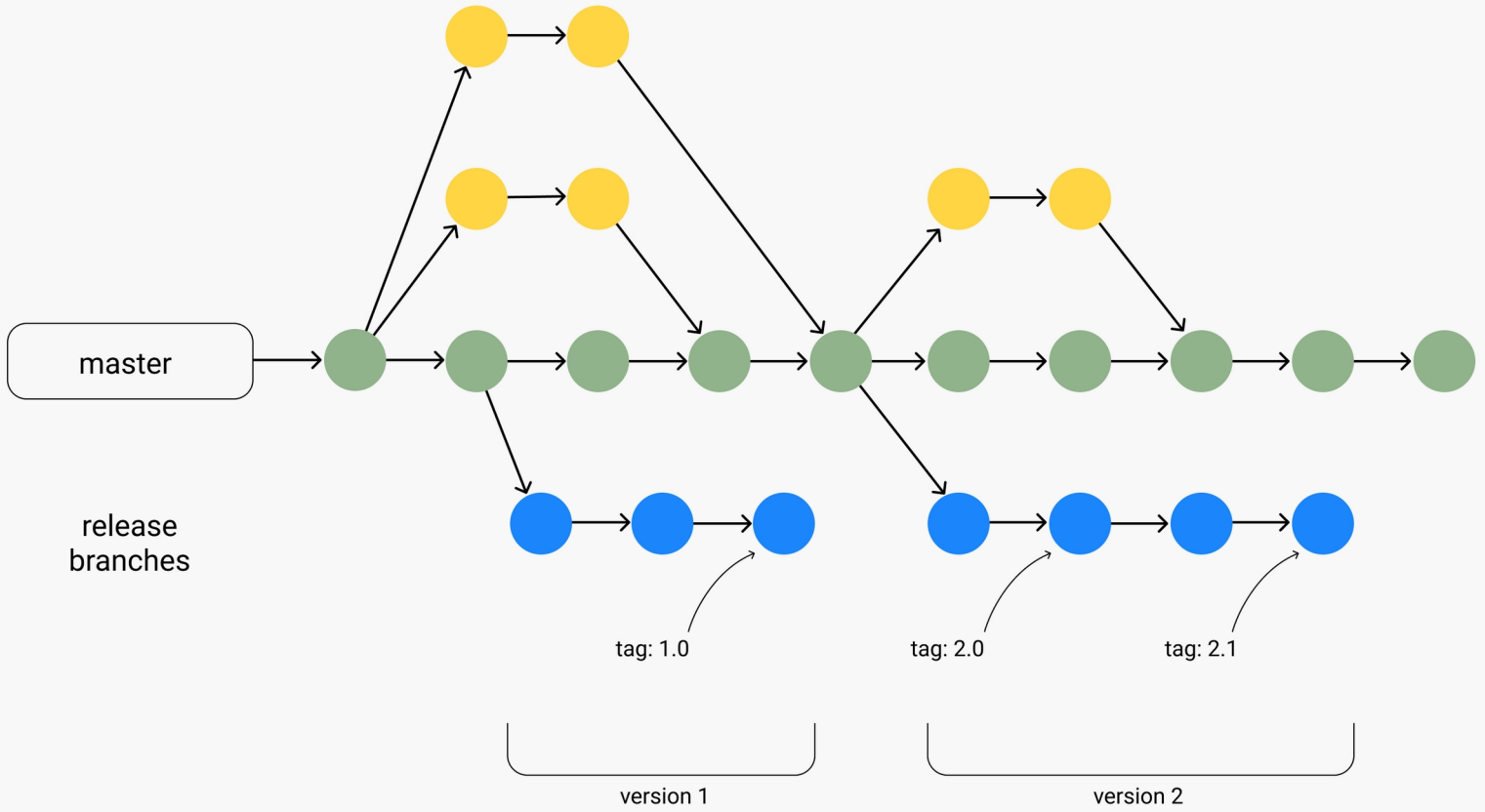


`git cherry-pick f` - take commit f to branch

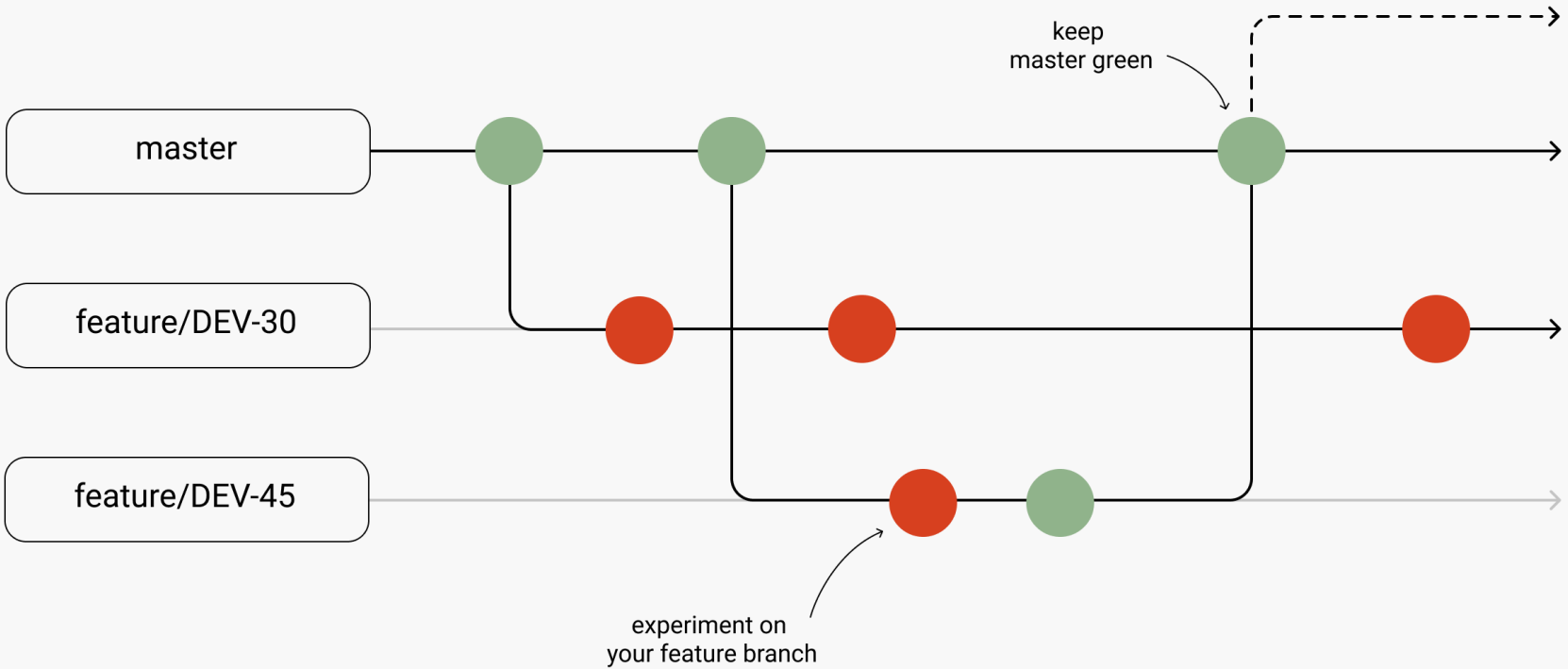


Branching strategies

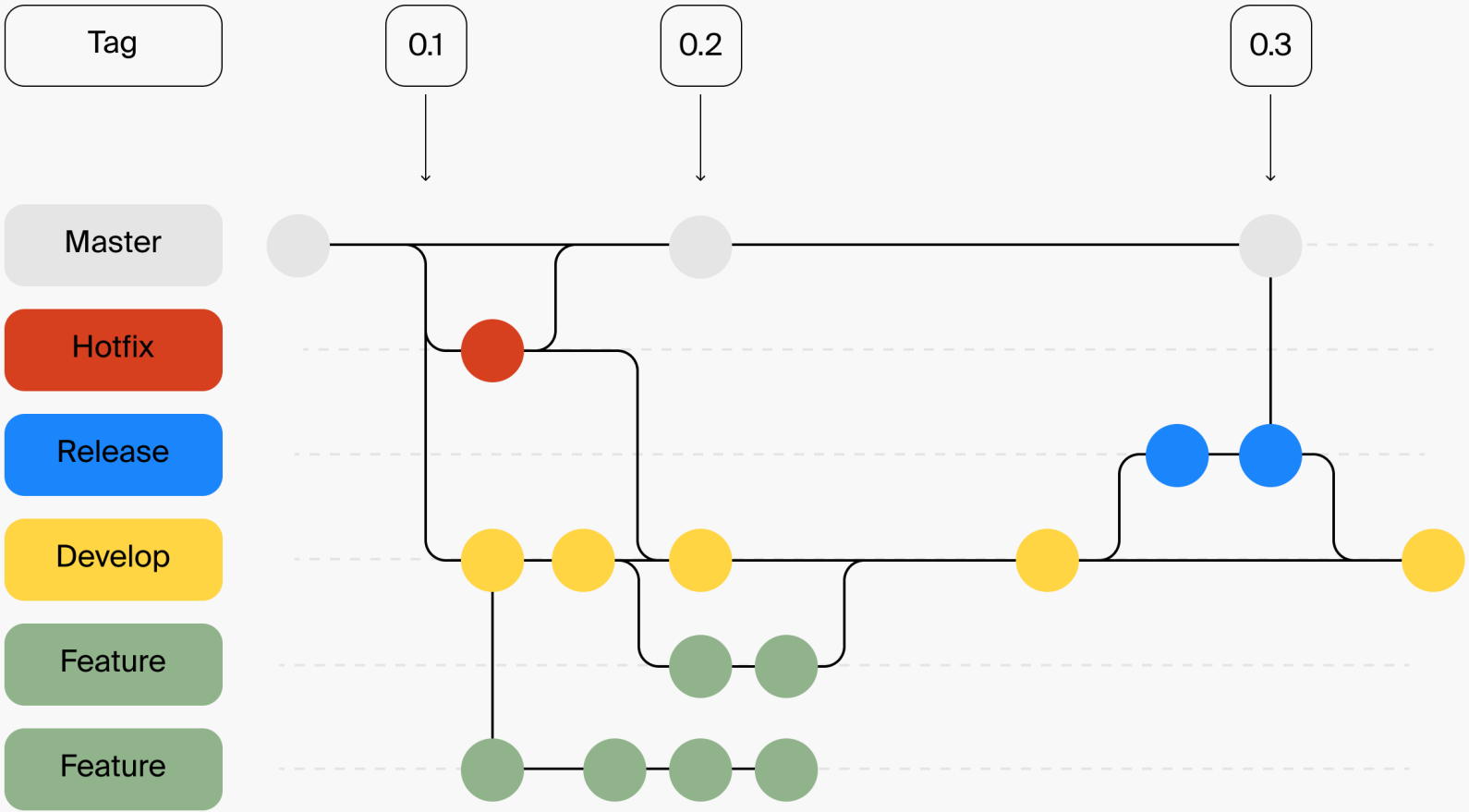
Trunk-based development



Feature branch workflow



Gitflow



Git best practices

- One change – one commit
- Merge is usually better than rebase
- When working with remote, don’t forget to pull common branches
- **git blame <file>** - check author of changes
- IDE usually has convenient git plugins