Git Workflow

http://git-scm.com

Anar Manafov, GSI Darmstadt
2014-03-21

Motivation

come up with a git workflow, which will provide

- uninterruptible development,
- a stable and releasable at any time master,
- a multilevel protection against conflicts,
- a delegation of conflicts resolution to authors,
- a multilevel possibility to recover from errors/mistakes before changes land into the master,
- a clean history of the master branch without merge commits and other unwanted garbage.

Requirements

- Contained development: Every task/feature MUST be implemented on a separate branch. Basically each JIRA ticket should be represented by at least one branch.
- Only release managers are allowed to work on the central master branch.
- Cherry picking MUST not be used by any means. A good use of branches should prevent the need of "git cherry-pick".
- DO NOT create very large repositories.
- DO NOT commit large binary files.
- DO NOT commit any file, which can be recreated or which is generated automatically by your dev. environment.

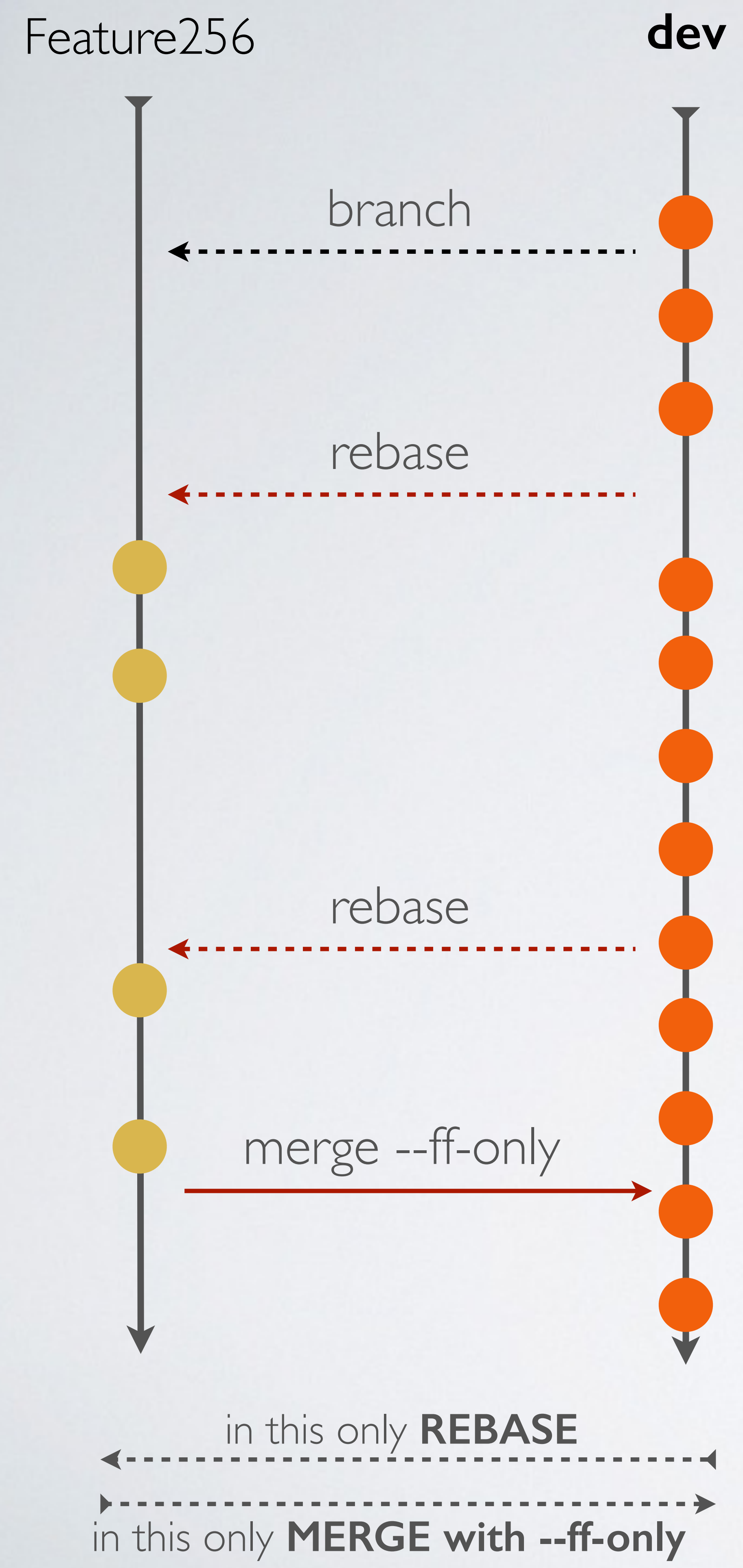
Branches

- master
- dev
- featureX
- RC
- HotFix

Roles

- Release manager (r/w: MASTER, DEV, RC)
- Developer (r/w: FEATURE(s), HOT FIX; read only: MASTER, DEV)

User Stories

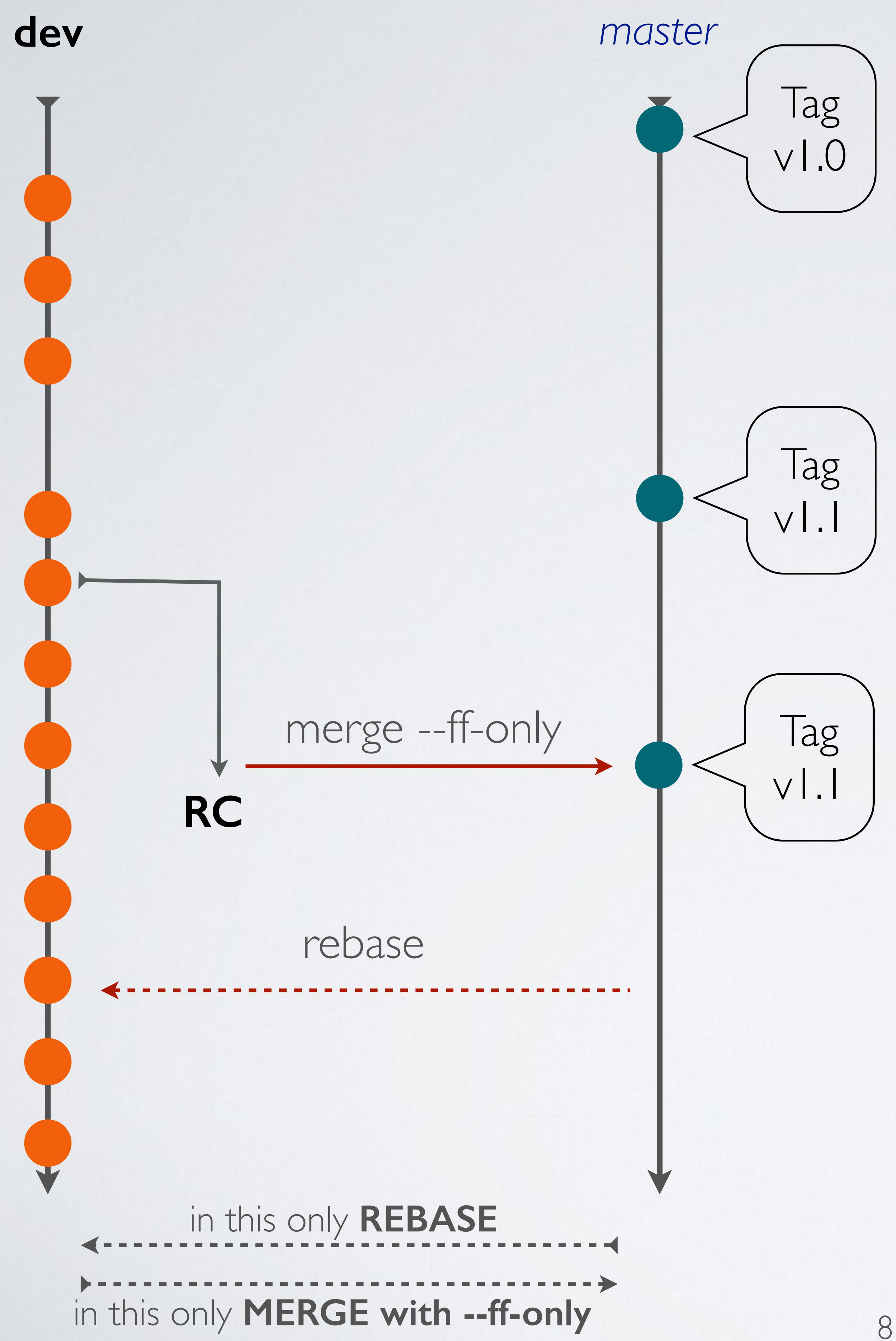


DEVELOPER

1. Create a feature branch;
2. Develop on it.
3. **Rebase regularly** from the dev branch.
4. **Rebase before** requesting to pull.
5. **Squash your commits** before requesting to pull.
6. Send a pull request.

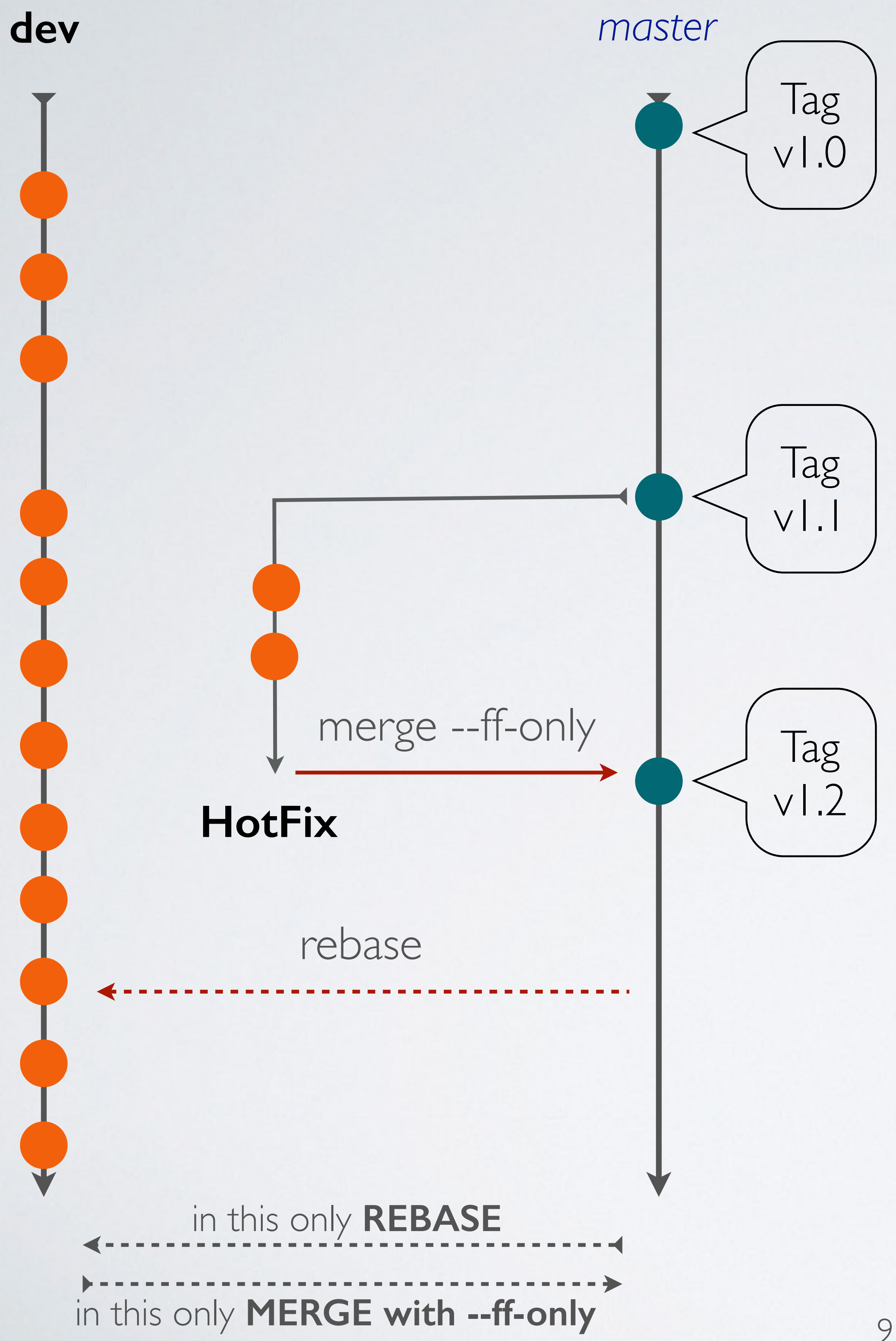
RELEASE MANAGER (can be a robot)

1. Receive the pull request (email, voice or any other method).
2. **Merge** the commit **using fast forward only**!
3. Git will abort merging if FF is not possible. In this case reject the commit and ask developer to rebase from dev and fix conflicts.



RELEASE MANAGER

1. Choose the last commit, which should be in the next release.
2. Create a Release Candidate branch from that commit.
3. Start prerelease QA.
4. RC is a feature freeze of the the product. Only bug fixes are allowed.
5. When QA gives the green light, merge the RC branch into the master with —ff-only.
6. Create a version tag based on a given master commit.
7. Dispose the RC branch



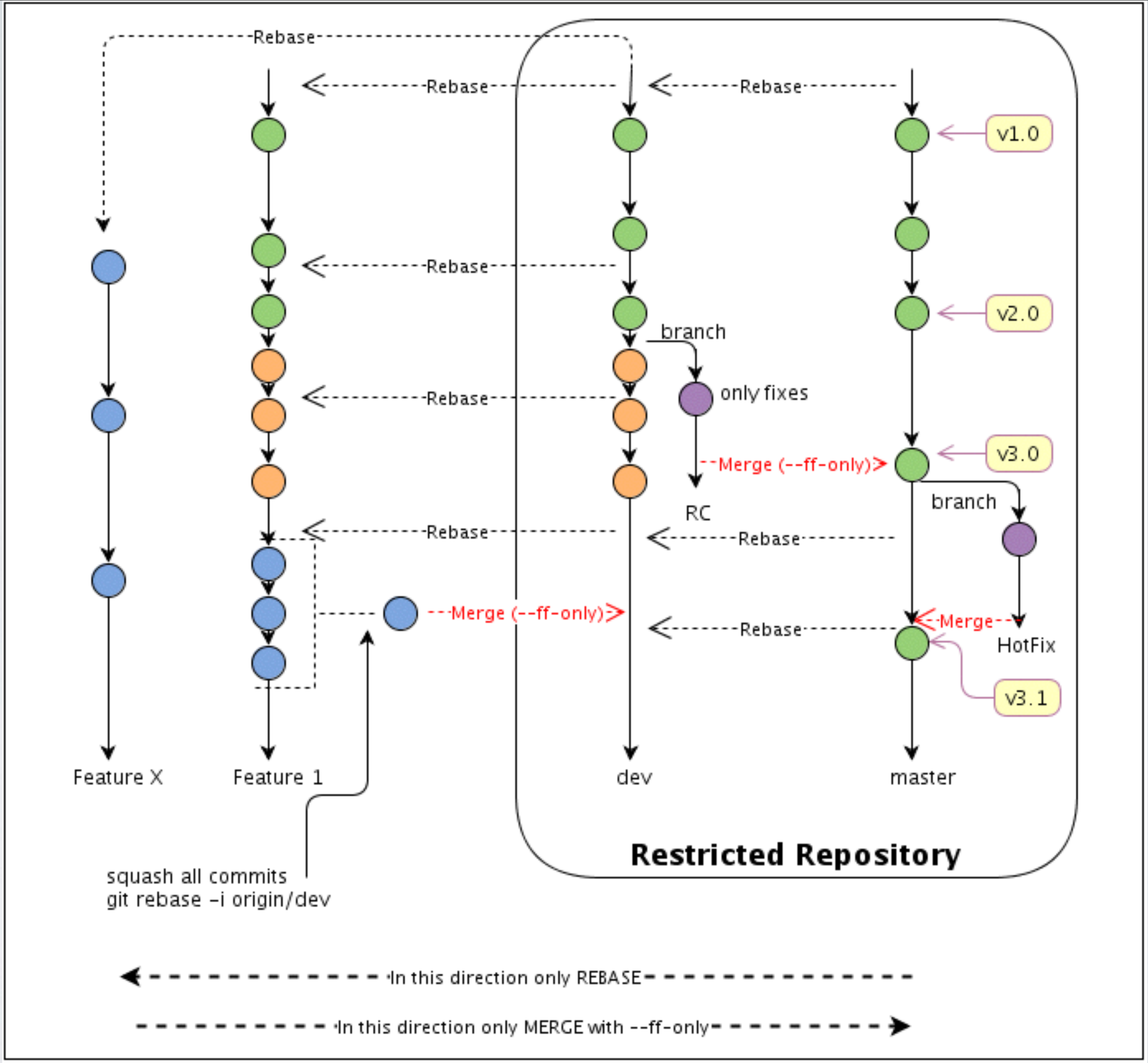
Note:

Here the workflow is the same like in case of usual development, just with one difference that developers work with the **HotFix** branch instead of **dev**.

At the end commits should be squashed and merged into the master with FF.

After that the master gets a new tag.

Don't forget to rebase the dev from the master so that developers will get applied fixes as well.



a detailed doc: http://fairroot.gsi.de/?q=node/83