Branching and Release Policy, GitFlow

Branching and release policy for UnifiedViews internal contributors

(Taken from http://nvie.com/posts/a-successful-git-branching-model/, where more details may be found)

All repositories contain two main branches with an infinite lifetime:

- **master** - the source code of `HEAD` always reflects a *production-ready* state.
- **develop** - the source code of `HEAD` always reflects a state with the latest delivered development changes for the next release.

When contributing to the project, **you must always create a temporal branch for you contribution**. Such branch must be deleted when properly merged back to develop/master branches.

GitFlow

A tool (Git extension), which supports this branching and release policy specified above, see http://danielkummer.github.io/git-flow-cheatsheet/ for details.

You as an internal contributor should always use this git flow extension if possible, because it simplifies the proper use of the branching and release policy.

For installation guide, please see http://danielkummer.github.io/git-flow-cheatsheet/#setup

Branches:

We distinguish the following types of branches. For each branch, we introduce:

1) who may create such branches

2) From which main branch (develop or master) the branch may be created

3) To which main branch (develop or master) the branch may be merged

We also introduce for each branch git flow command, which should be used.

Feature branches

Who may create such branches: Internal contributors, Repository managers

May branch off from: develop

Must merge back into: develop

Feature branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. When starting development of a feature, the target release in which this feature will be incorporated may well be unknown at that point. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).

Unless the feature branch is shared by more developers, feature branches typically exist in developer repos only, not in origin (it is not mandatory to push /publish feature branch into github repository).

Guideline:

1. Creates new feature branch feature/{featureName} based on the develop branch and switches to that branch
   a. git flow feature start {featureName}
2. (Optional) If more internal contributors should collaborate on the release, you need to publish the feature branch
   a. git flow feature publish {featureName}
3. Work on that branch as needed
4. Merges feature/{featureName} branch back to the develop branch, removes the feature branch, switches to develop branch (⚠️ rebase shall not be used)
   a. git flow feature finish {featureName}

Release branches

Who may create such branches: repository managers. Internal contributors may only contribute to the release branched when invited, they may not create release branches

May branch off from: `develop`
Must merge back into: `develop` and `master`

Release branches support preparation of a new production release. They allow for last-minute dotting of i’s and crossing t’s. Furthermore, they allow for minor bug fixes and preparing meta-data for a release (version number, build dates, etc.). By doing all of this work on a release branch, the `develop` branch is cleared to receive features for the next big release.

Adding large new features here is strictly prohibited.

See more details here: Releasing & Hotfixing UnifiedViews

Hotfix branches

Who may create such branches: Internal contributors, repository managers

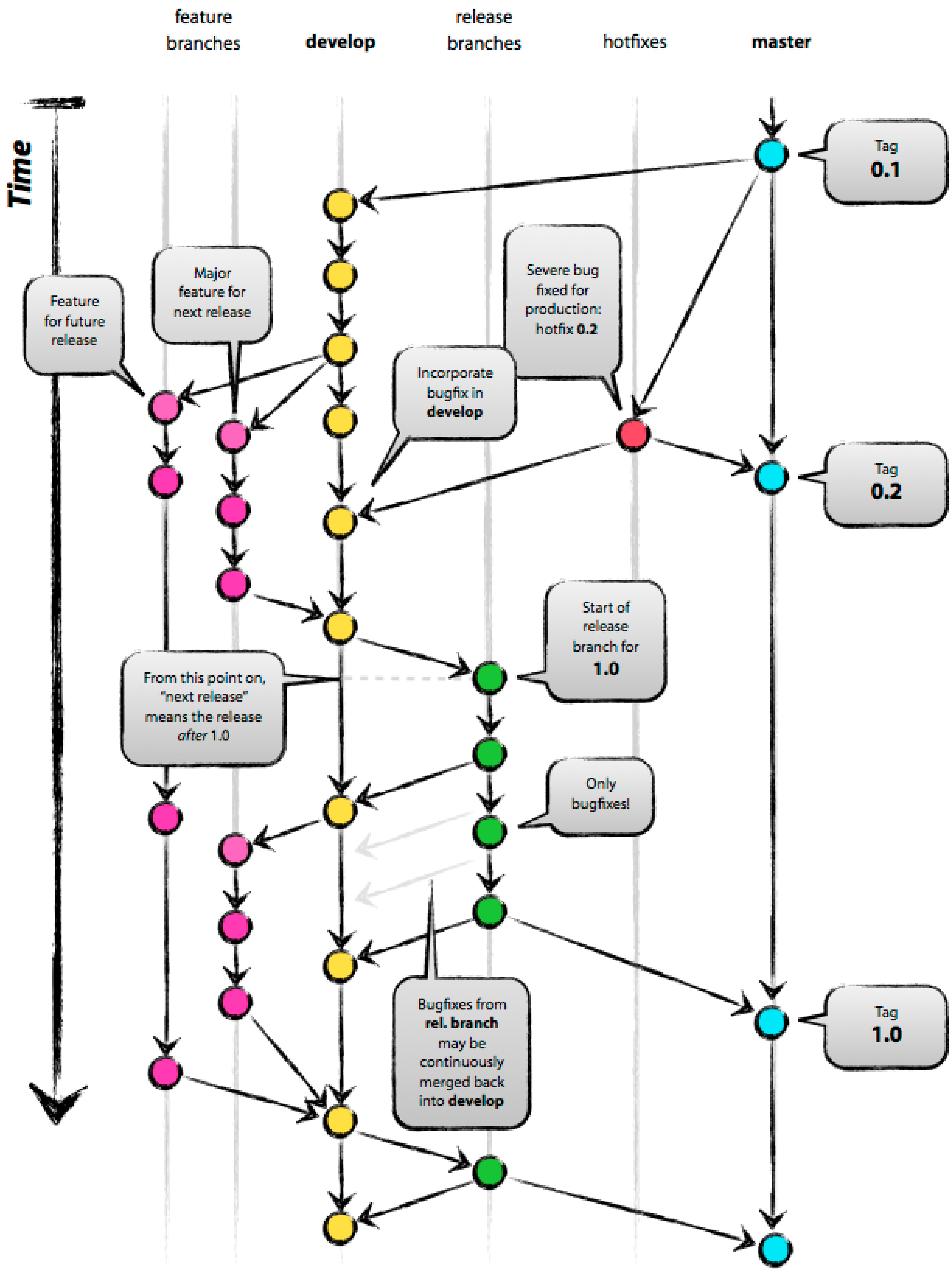
May branch off from: `master`
Must merge back into: `develop` and `master`
Branch naming convention: `hotfix/{name}`

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

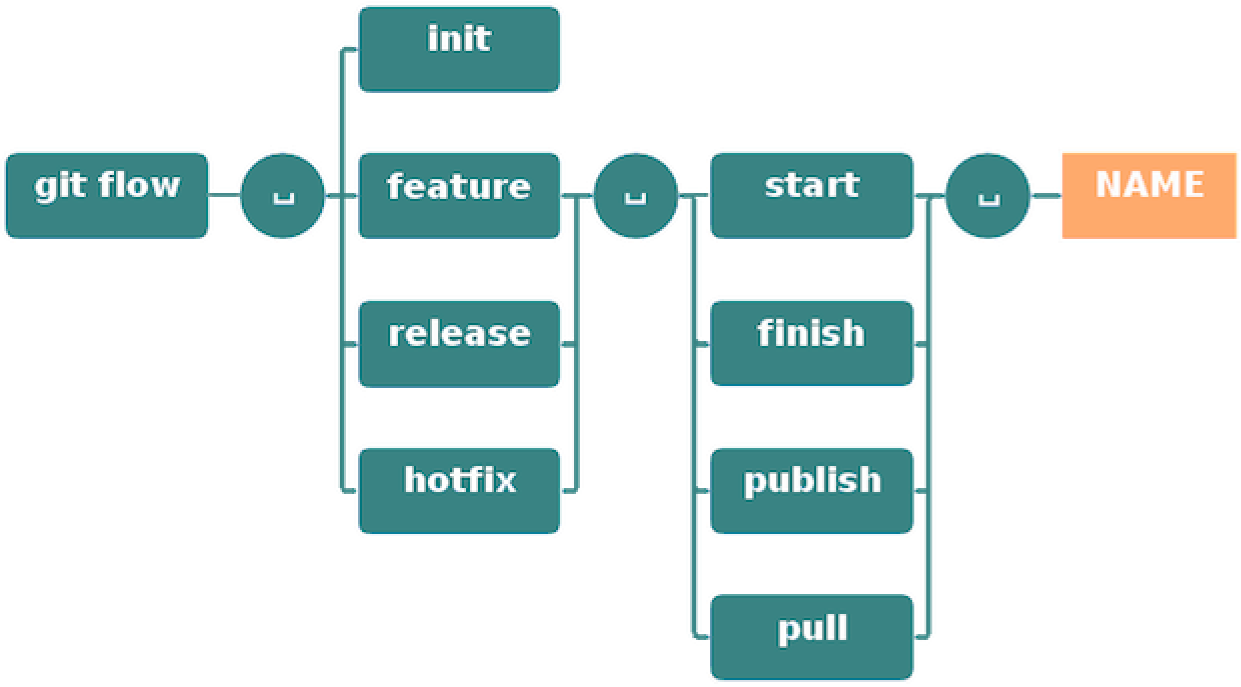
See more details here: Releasing & Hotfixing UnifiedViews

Summary

Summary of all types of introduced branches:


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Summary overview of the git flow commands available:


git flow
␣
init
feature
release
hotfix
␣
start
finish
publish
pull
␣
NAME