GIT BRANCHING GUIDELINES

The following document describes in details company-wide guidelines for working with branches in a software project

GLOSSARY

PTL	Project Technical Lead
Feature	The set of the bounded functionality
Bug fix	Defect fixes for the prepared release
Hot-fix	Bug fix for the production release
Release	The set of the stable features including bug fixes or hot-fix
Main branches	Branches which are required to be in the central remote repository and have infinite lifetime. Any of the main branches should be created by PTL on the start of development (or any developer who starts the project) and should be marked as protected to: - prevent pushes from everybody except masters - prevent anyone from force pushing to the branch - prevent anyone from deleting the branch
Supporting branches	Any branches which need to aid parallel development between team members, ease tracking of features, prepare for production releases and to assist in quickly fixing live production problems. Unlike the main branches, these branches always have a limited life time, since they will be removed eventually.
Master branch	Origin/master is the main branch where the source code of HEAD always reflects a production-ready state.
Development branch	Origin/development is the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release.
Feature branch	Feature branches are used to develop new features for the upcoming or a distant future release.
Release branch	Release branches support preparation of a new production release. 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.
Hotfix branch	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.

BRANCH NAMING

The following tables describe branch naming and responsible actors for each of the branch types

PRIMARY BRANCHES

Branch type:	Master
Branch naming convention	master
Created By:	PTL (or any developer who starts project)
Pushing By:	PTL
May branch off from:	-
Must merge back into:	-
Description:	Origin/master is the main branch where the source code of HEAD always reflects a production-ready state.
Additional requirements:	-
Auto-Build:	No

Branch type:	Development
Branch naming convention	development, develop
Created By:	PTL (or any developer who starts project)
Pushing By:	PTL
May branch off from:	master
Must merge back into:	master
Description:	Origin/master is the main branch where the source code of HEAD always reflects a production-ready state.
Additional requirements:	When the source code in the development branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number.
Auto-Build:	Yes

SUPPORTING BRANCHES

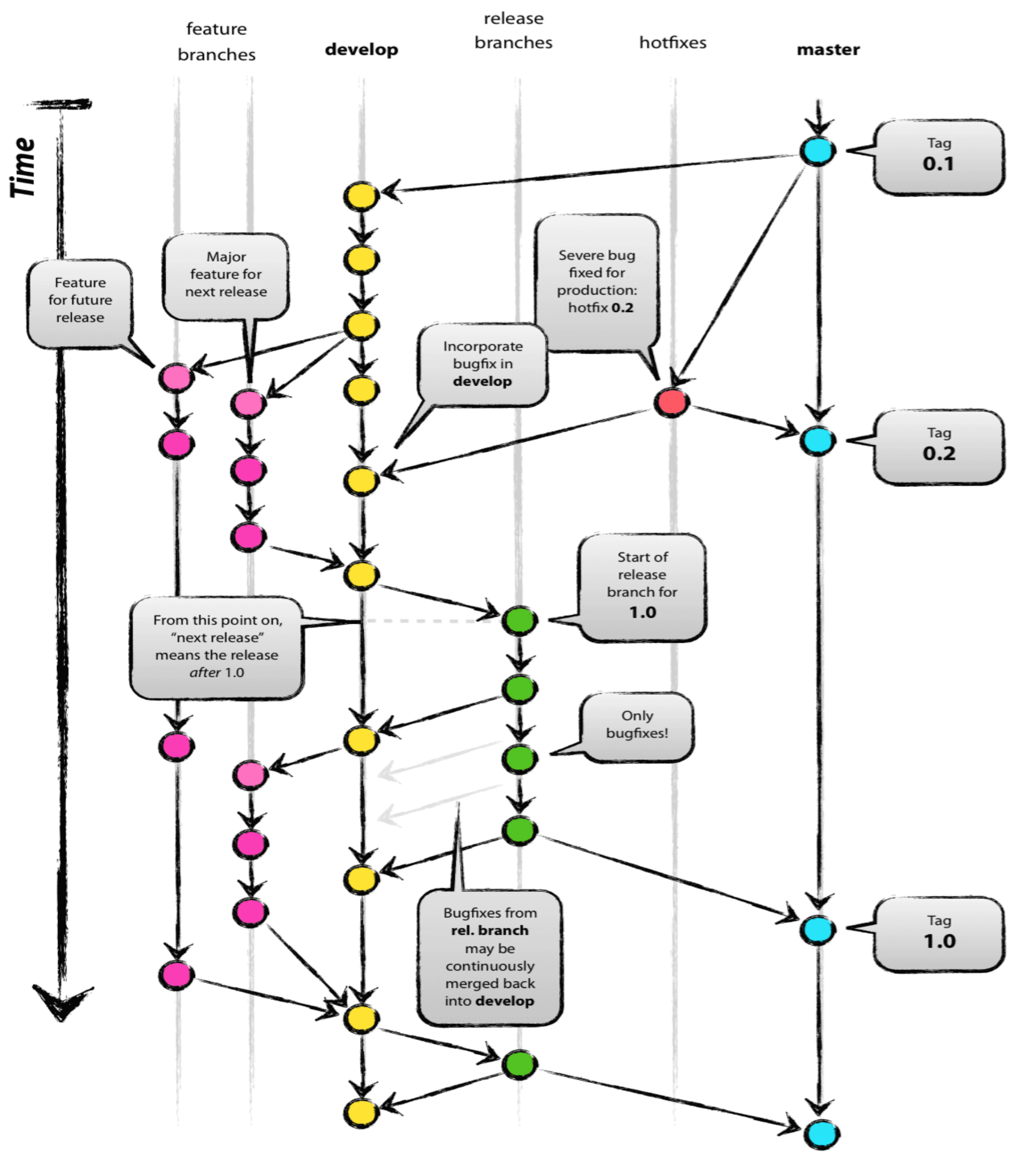
Branch type:	Feature
Branch naming convention	feature<N>, where N is number of the feature in the tracking system
Created By:	any developer
Pushing By:	any developer
May branch off from:	development
Must merge back into:	development (by PTL only)
Description:	Feature branches (or sometimes called topic 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 development (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).
Additional requirements:	When developer finished feature development he should push it to the remote branch with the same name and send pull-request to the development branch
Auto-Build:	No

Branch type:	Release
Branch naming convention	release<N>, where N is version number
Created By:	PTL (or any developer who starts project)
Pushing By:	PTL
May branch off from:	development
Must merge back into:	development and master (by PTL only)
Description:	Release branches support preparation of a new production release. 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. The key moment to branch off a new release branch from development is when development (almost) reflects the desired state of the new release. At least all features that are targeted for the release-to-be-built must be merged in to development at this point in time. All features targeted at future releases may not—they must wait until after the release branch is branched off.
Additional requirements:	This branch should be created only if we want to support particular release version
Auto-Build:	-

Branch type:	Hotfix
Branch naming convention	issue<N>, where N is defect number in the tracking system
Created By:	any developer
Pushing By:	any developer
May branch off from:	development
Must merge back into:	development and master (by PTL only)
Description:	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.
Additional requirements:	The pushing process the same as the feature branches.
Auto-Build:	No

The next workflow illustrated branches and development process:



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

PULL REQUESTS

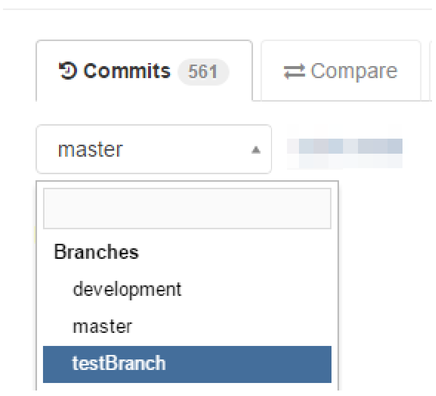
Pull requests let you tell others about changes you've pushed to a repository on GitHub. Once a pull request is sent, PTL can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary. These changes are proposed in a branch, which ensures that the main branches are kept clean and tidy.

Pull requests can only be opened if there are differences between your branch and the upstream branch. You can specify which branch you'd like to merge your changes into when you create your pull request.

The following section describes steps necessary to create pull request.

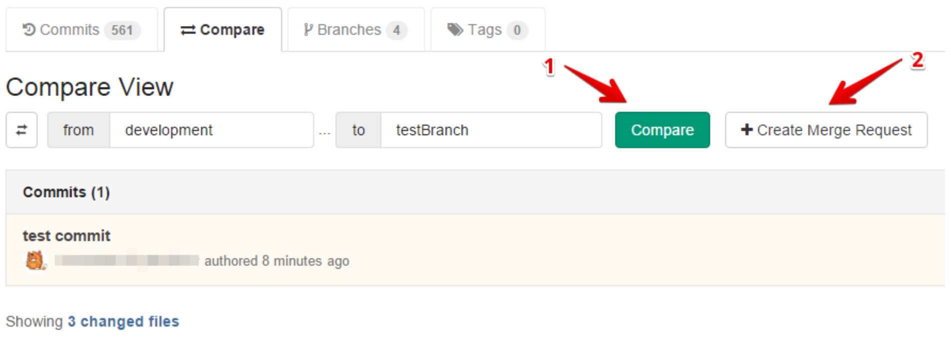
1) On GitLab, navigate to the repository from which you'd like to propose changes.

2) In the "Branch" menu, choose the branch that contains your commits.



3) Select the Compare Tab - the Compare page will automatically select the base and compare branches; you can change these, click “Compare”. You will see all the changes, commits etc.

4) On the Compare page, click Create Merge Request.



Commits (1)

test commit

authored 8 minutes ago

Showing 3 changed files

5) Type a title and description for your pull request.

6) Click Submit new merge request.

After your pull request has been reviewed, it can be merged into the repository.

TYPICAL TASKS

There are a number of typical tasks while development process is taken:

- Implementation of the new feature
- Preparing release
- Resolving a defect
- Resolving production issue (hotfix)

The following section provides list and examples of each of them

IMPLEMENTATION OF THE NEW FEATRE

Step	Example
1. Update development branch	> git checkout development > git pull origin development
2. Create feature local and remote branch	> git checkout -b feature666 > git push -u origin feature666
3. During feature development developer should commit every completed task of the feature	> git add . > git commit -a -m “6666: Add item updating”

	`> git push origin feature666`

4. Before creating pull request responsible developer should verify that merge into development branch will be a fast-forward	*`# get the latest development version`*
	`> git checkout development`
	`> git pull origin development`
	
	`> git checkout feature666`
	`> git rebase development`
	
	# fix conflicts if it needed, also merge command can be used
	
	`> git push origin feature666`

5. After finishing feature development pull request should be created by responsible developer	-

6. When pull request is confirmed by PTL and merged into main branch, responsible developer has to remove both local and remote issue branches	`> git branch -d feature666`
	`> git push origin :feature666`
	
	`# Since Git v1.7.0 you can use the following command`
	`> git push origin --delete feature666`

PREPARING RELEASE

PTL is responsible for feature branch review and merging it into development branch if review was passed

When development branch has enough features to build new release PTL should merge it to master branch

Step	Example

1. Update development branch	`> git checkout development`
	`> git pull origin development`

2.1 Merge it to master branch	`> git checkout master`
	`> git merge development`
	
	# it will be useful to keep branch history and avoid fast-forward algorithm while merging with command
	
	`> git merge --no-ff development`

2.2. Alternatively, PTL can merge branches using rebase command to follow linear commit history (instead p.2.1)	`> git checkout development`
	`> git rebase master`
	`> git checkout master`
	`> git merge development`

3. Create a new tag with the next release version	`> git tag -a v1.4 -m "release version 1.4"`
4. Push updates to remote branch	`> git push --follow-tags origin master`

RESOLVING A DEFECT

Step	Example
1. Get the latest updates from the development branch	`> git checkout development` `> git pull origin development`
2. Create issue branch	`> git checkout -b issue888`
3. After bug is fixed responsible developer should push updates to remote	`> git checkout development` `> git pull origin development` `> git checkout issue888` `> git rebase development` `> git push origin issue888`
4. Create pull request	-
5. When pull request is passed, responsible developer has to remove both local and remote issue branches	`> git checkout development` `> git branch -d issue888` `> git push origin --delete issue888`

RESOLVING PRODUCTION ISSUE (HOTFIX)

Step	Example
1. Update master branch	`> git checkout master` `> git pull origin master`
2. Create and switch into hotfix branch	`> git checkout -b issue999`
3. Fix bug and push it to remote branch	`> git add .` `> git commit -a -m “999: fix ugly button”` `> git push -u origin issue999`
4. Create pull request to **master** branch	-
5. When pull request is passed, responsible developer has to remove both local and remote issue branches	`> git checkout development` `> git branch -d issue999` `> git push origin --delete issue999`
PTL RESPONSIBILITIES	
6. Review pull request	-
7. Merge it into both master and development branch	`> git checkout --track origin/issue999` `> git checkout development` `> git merge issue999` `> git checkout master` `> git merge issue999`

8. Create tag with new PATCH version and push updates	> git tag -a v1.4.1 -m "patch version 1.4.1" > git push origin development > git push --follow-tags origin master
9. Remove hotfix branch	> git branch -d issue999 > git push origin –delete issue999