

GIT process

Note: page moved to <https://confluence.eurodir.eu/confluence/display/KlantenVertellen2/GIT+process>

- Required GIT configuration settings
 - Branches
 - Develop and create pull requests
 - Merging the pull request
- Release
 - Release candidates
 - Versioning scheme
 - Releasing
 - Create "<major>.<minor>" release branch
 - Create a release candidate
 - Work on a release branch
 - Deploy release candidate to PRODUCTION
 - Merge the release branch
 - Create hotfixes
 - Release diagram
 - Tips
 - Misc

Required GIT configuration settings

To minimize the potential for problems, please run the following GIT commands. They will configure GIT to to use safe settings globally (for all repositories):

```
git config --global push.default simple
git config --global rerere.enabled true
```

The option "push.default" insures that GIT will not push multiple branches when executing "git push", but only the current branch. Without this configuration option, if used with "--force" or "--force-with-lease", git will push all branches, and because of the "--force" it will override remote history with potentially older history, losing some commits. The option is good to have in any case (and is the default for GIT 2.0), but is especially critical for the workflow described here, because we need to use "--force-with-lease".

If your git client doesn't support the "--force-with-lease" option, you are using a really old version. In this case, please upgrade to the latest version.

To check if your client supports this option, run "git help push" and check if the option is described on the help page.

The option "rerere.enabled" insures that we don't have to solve the same conflicts multiple times during a rebase. The option is needed because the workflow described here uses rebase. This enables the [rerere git feature](#), and since it's a configuration setting, you only need to do it once. Of course, if you re-install your OS, you need to do it again.

Development

Branches

- "master":
 - This is the "integration" branch, into which we merge feature branches. The latest code is here.
- "release/<major>.<minor>":
 - This is the branch used to release the version "<major>.<minor>" from the master branch.
 - We will re-use this branch for all hotfixes also, rather than having separate branches.
 - We will merge this branch into "master" repeatedly - after each hotfix release.
- feature branches

Develop and create pull requests

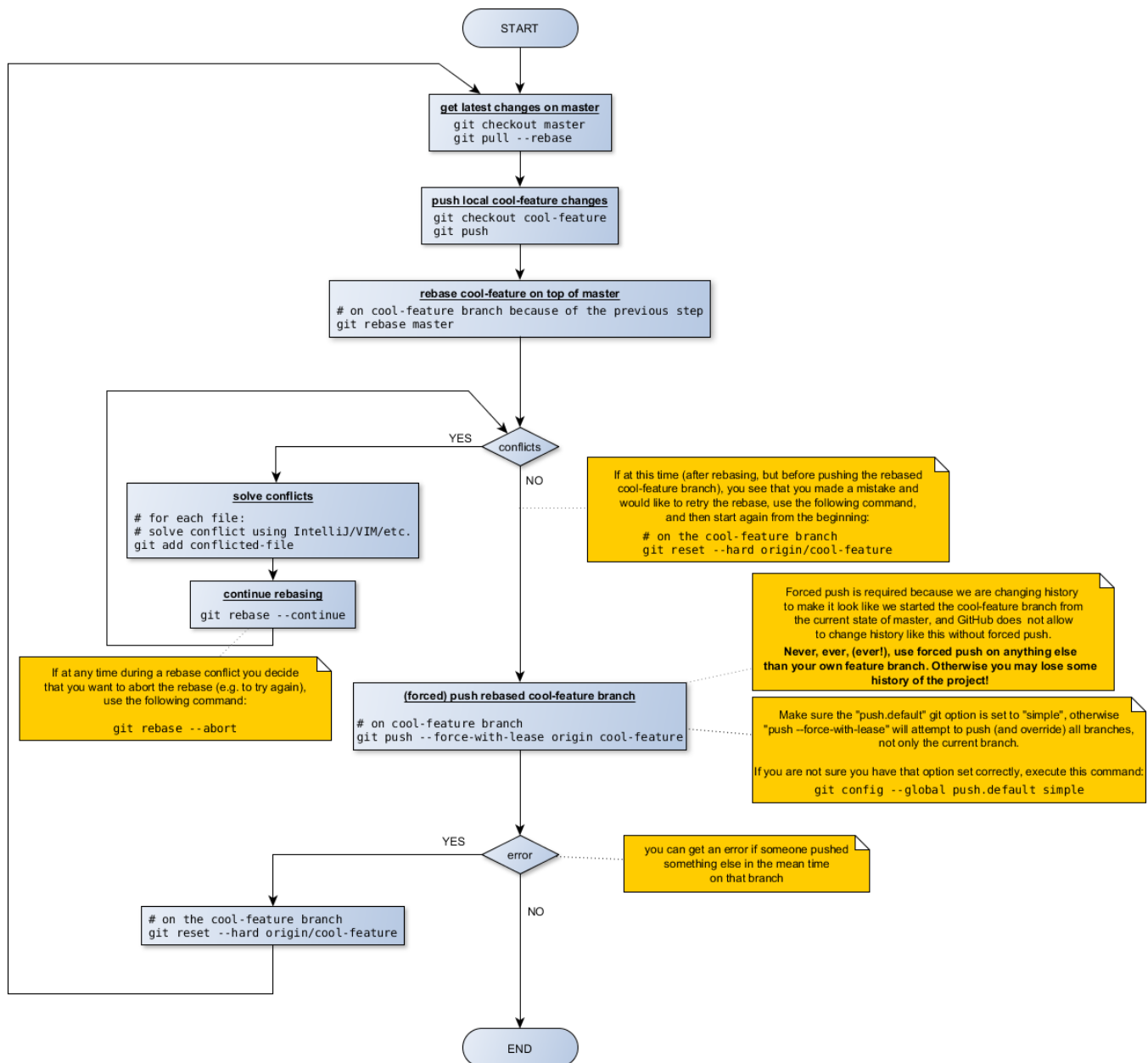
This section will explain a practical example of development, using examples of git commands.

When you start development of a new feature, bug fix, or other piece of work, you **create a new branch from master**, first making sure it's updated from the remote:

```
git checkout master
git pull --rebase
git checkout -b cool-feature
git push -u origin cool-feature
```

You have now created the new branch from `master`, and pushed it to Github. You can delay pushing until after you have done some work, but remember to **push often**, to avoid losing work.

Let's say the next day you want to get into your feature branch the other changes which were merged into master in the mean time. In this case, you should **rebase your feature branch on top of master**:



Only rebase after pushing all your local changes to GitHub. This way, you make sure you don't lose your code if you make a mistake locally during rebase.

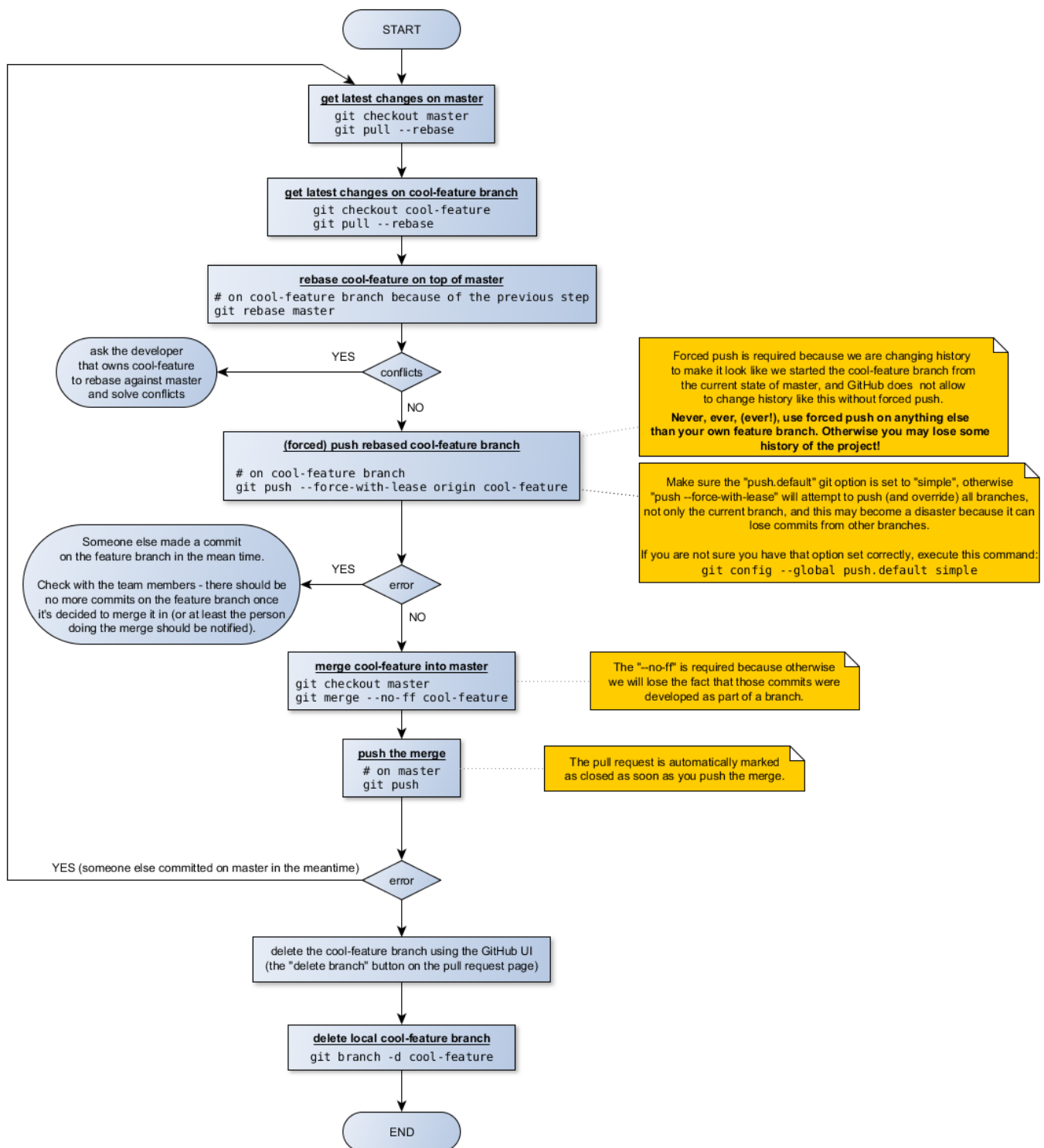
Only use --force-with-lease to push your feature branch. Never, ever (EVER!), use it on any other branch (unless you really know what you are doing) as you may destroy existing history.

Once you are done with your feature **create a pull request** on Github.

After creating the pull request, move the associated JIRA ticket from status "In progress" to the status "Code review".

Merging the pull request

Now, other team members will **check the pull request, provide feedback**, and possibly **point out defects** in the code. **Defects have to be fixed** before the pull request may be merged. Once the pull request is approved, it should be **merged to master**:



After the pull request is merged and the merge pushed (in other words, after the process in the above diagram is done), please move the associated JIRA ticket from "Code review" status to "Ready for deployment".

Release

Release candidates

It's a best practice to deploy the same exact binary in ACCEPTANCE and PRODUCTION. For this reason, we will only deploy releases produced

by the maven release plugin (no snapshots). We will call these builds "release candidates".

Because there is a chance that a release candidate will be rejected after testing in ACCEPTANCE, we may have multiple "release candidates" for a single release, until one of them is accepted.

The release candidate that passes testing in ACCEPTANCE will be deployed to PRODUCTION.

Versioning scheme

The versioning scheme is "`<major>.<minor>.<hotfix>.<release-candidate>`".

All fields are required.

They have the following meaning:

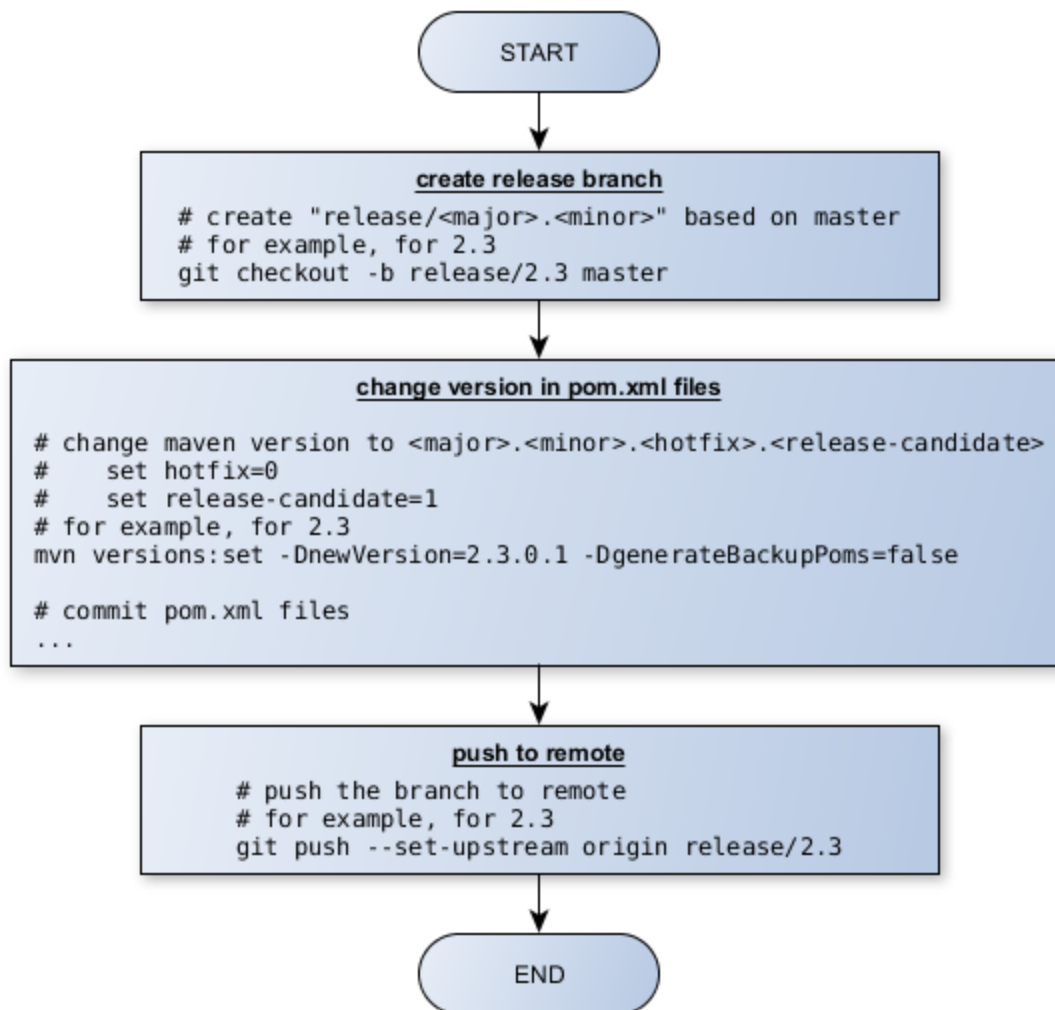
- "`<major>.<minor>`" is the business version
- "`<hotfix>`" is used because we can have multiple releases for the same "`<major>.<minor>`" version.
 - it starts with "0"
 - the first release is "hotfix 0". Hotfix releases after this will have this field incremented, so that the first hotfix will have `hotfix=1`.
- "`<release-candidate>`" is used to track the release candidate number.
 - it starts at "1"
 - ideally, this will always be "1". In real life, a release candidate can be rejected by testing in ACCEPTANCE. If a release candidate is rejected after testing in ACCEPTANCE, we fix the problems, then increment this field, and create a new release candidate to be tested in ACCEPTANCE. We do this until a release candidate is accepted.
- on the "master" branch, the version will be always "`master-SNAPSHOT`". This makes things easier: we don't need to track the version number on this branch, since we release from a separate branch. We will track the version number on the release branch.

Releasing

Create "`<major>.<minor>`" release branch

When the "master" is considered ready to release, we need to create a release branch using the following steps

- create the release branch and push it to GitHub ("`release/<major>.<minor>`").
- change the version in the pom.xml files to "`<major>.<minor>.0.1-SNAPSHOT`" (`hotfix=0`, `release-candidate=1`), commit, and push to GitHub



This is not needed for hotfix releases, since for hotfixes we will reuse the current release branch. Refer back to the [Branches section](#).

Create a release candidate

Creating a release candidate should only be done from a release branch ("release/<major>.<minor>").

To create a release candidate, just use the [Jenkins release job](#), with the following parameters:

- release branch: this is the name of the branch from which to release, for example "release/2.3". This needs to be a release branch, not "master".
- version to release: "<major>.<minor>.<hotfix>.<release-candidate>" (same as the current version in pom.xml, without "-SNAPSHOT")
- next development version: "<major>.<minor>.<hotfix>.<release-candidate + 1>-SNAPSHOT" (increment the <release-candidate>)

If successful, the release job will upload the artifacts on Nexus.

After the Jenkins build is done, merge the release branch into master, for example:

```
# pom.xml files were changed by Maven in the Jenkins build; get them
git checkout release/2.3
git pull --rebase origin release/2.3

# make sure master is also up-to-date
git checkout master
git pull --rebase origin master

# merge
# the "--no-commit" flag will instruct git not to automatically commit the
merge
git merge --no-ff --no-commit release/2.3

# manually revert the version changes in the pom.xml files, to keep the
version at "master-SNAPSHOT"
...

# commit the merge
git commit

# push new master that contains the merge of the release branch
git push origin master
```

Once a release candidate is created, it will be deployed on ACCEPTANCE and tested. If it passes testing on ACCEPTANCE, it will become the release. We will take the same maven artifact that we deployed on ACCEPTANCE and we will deploy it to PRODUCTION.

If the release candidate is rejected after testing on ACCEPTANCE, we need to:

- do the necessary code fixes (see the section "Work on a release branch")
- create a new release candidate and deploy again on ACCEPTANCE. This one will have an incremented <release-candidate> field in the version because that is what we set for the next development version, when we created the previous release candidate (see above).

Work on a release branch

Issues are fixed using pull requests, just as we do on the "master" branch.

Deploy release candidate to PRODUCTION

After a release candidate has passed testing on ACCEPTANCE, we will deploy it also on PRODUCTION.

After deployment to PRODUCTION, if this is a new "<major>.<minor>" version (not a hotfix), we need to delete the release branch associated with the old version. For example, after we deploy, the version 2.3, we need to delete the branch "release/2.2". We do this, because we need only one active release branch - the one corresponding to the current PRODUCTION release. We need it in case we need to create a hotfix.

To get back to old releases, we have the tags created by the maven release plugin.

Merge the release branch

After creating a release candidate, the release branch needs to be merged into "master".

After merging, please make sure that the maven version in pom.xml files is still "master-SNAPSHOT".

Create hotfixes

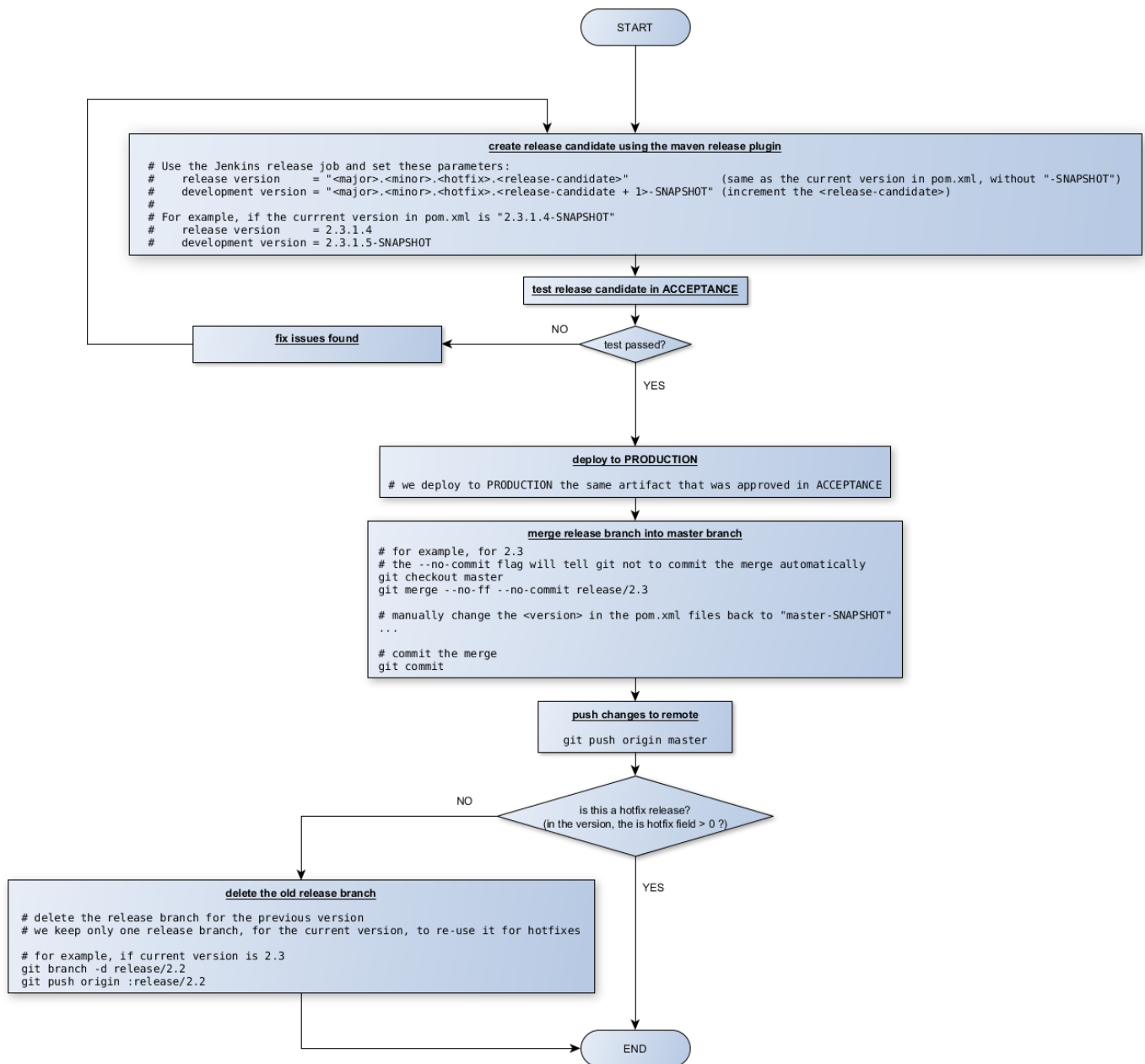
Hotfixes are created on the release branch corresponding with the current release. No new branch is needed.

To create a hotfix:

- change version in pom.xml files
 - increment <hotfix>
 - set <release-candidate> to 1
 - example: 2.7.2.3 will become 2.7.3.1
- fix the issue in a feature branch
- create a pull request
- merge the pull request into the release branch

Release diagram

The following diagram summarizes the release process.



Tips

Many of the procedures above requires to change the version in the pom.xml files. To do it safely and easily, please use the "versions" maven plugin so:

```
mvn versions:set -DnewVersion=<the-version-we-need-to-set>  
-DgenerateBackupPoms=false
```

Misc

The diagrams on this page were created using the program [yEd](#). In case you need to modify the diagrams, you can find the source code (*.graphml files) attached to this page (look in the list of attachments).