This document describes how the Spring Framework team manages Git branches and how changes are applied to multiple branches. For more details about the code conventions, check out the [[Code Style]] page.

Git Branches

When looking at the Spring Framework repository, we can find several branches:

- the current branch - the branch of the current Framework generation, i.e. the latest generation with GA releases. This information is available on the Spring Framework project page
- the Git `master` branch - this can be the current branch, or a branch dedicated to the next major/minor release
- maintenance branches - branches of Framework generations being actively maintained
- former maintenance branches of EOL'd versions, i.e. not maintained anymore

Applying changes to the Spring Framework repository

For this example, we'll use the following as the current state of the repository:

- the `master` branch is dedicated to the upcoming `5.3.0.RELEASE` minor version
- the current branch is `5.2.x`
- there are two active maintenance branches, `5.1.x` and `5.0.x`

Let's consider the issue `#1234`, which is the bug fix we're working on.

Forward merges

First, we need to fix the bug and merge it forward if necessary. If the current branch is the master branch, pushing changes to the master branch is enough and there is no need for forward merges. This means that we need to target the fix to the appropriate milestone, in this case the next `5.1.x` release.

```
$ # checkout the maintenance branch and update it
$ git checkout 5.2.x && git pull
$ # work on a fix and commit it to the current branch
$ git add . && git commit
$ # push the changes to the current branch
$ git push origin 5.2.x
$ # checkout the master branch and update it
$ git checkout master && git pull
$ # merge the changes forward, resolving conflicts if necessary
$ git merge --no-ff 5.2.x
```

Backports

If the fix is meant to be applied to other maintenance versions, you need to backport the commit to maintenance branches. The Spring Framework team is using the backport-bot for that.

To backport the fix for the issue `#1234`, we can directly cherry-pick the commit and push it to the branch. Doing so will automatically create a backport issue against the next 5.1.x release.

```
$ git checkout 5.2.x && git log
$ # the commit sha for the fix is c0ffee456
$ git checkout 5.1.x
$ git cherry-pick c0ffee456
$ git push origin 5.1.x
```

If you wish to create the backport issue beforehand, you can always tag the main issue with the `for: backport-to-5.1.x` GitHub issue label; the backport-bot will create an issue accordingly. Pushing the cherry-picked commit will close the backport issue, even if the commit message refers to the original issue.