This page serves as a simple "how to" for Spring committers when manually merging pull requests from contributors.

NOTE: Committers using Chrome or Firefox will want to install Phil Webb's handy user scripts that turn all SPR-1234 style JIRA ID's into a link and embed a div into the GitHub UI for all spring-framework pull requests containing the commands below, customized to the specific pull request in question.

Placeholders

Many of the `git` commands used on this page contain placeholders as defined below.

- `<ACCOUNT>`: GitHub account for the author of the pull request
- `<BRANCH>`: branch in pull request author's account (e.g., SPR-####)
- `<PULL_REQUEST_NUMBER>`: spring-framework pull request number

Example

To determine the values for these placeholders, let's take a look at pull request #111 as an example. From an *open* pull request page you should be able to find an example `git` command for pulling the request into your local working directory. For example, if you click on the *info* icon in the "This pull request can be automatically merged" bar, you should see something like: `git pull git://github.com/sbrannen/spring-framework.git SPR-9492`

From this information we determine the placeholder values to be the following.

- `<ACCOUNT>`: sbrannen
- `<BRANCH>`: SPR-9492
- `<PULL_REQUEST_NUMBER>`: 111

Performing the Merge

Set up remote, fetch branch, and rebase

```
$> git remote add <ACCOUNT> https://github.com/<ACCOUNT>/spring-framework.git
$> git fetch <ACCOUNT>
$> git checkout --track <ACCOUNT>/<BRANCH> -b <BRANCH>
$> git rebase master
```

Modify working directory

- polish and format the contribution as necessary, in line with the [[Contributor guidelines]]

- refactor code as necessary to comply with the [[Spring Framework Code Style]] (i.e., aim for uniformity with existing code, naming conventions, etc.)
- update and/or add Javadoc and reference manual documentation
- once changes are finalized and committed to the local branch, you typically will want to squash multiple commits into a single commit – for example, using `git rebase --interactive --autosquash` – however, sometimes it gives a more complete picture of the work that was performed in you leave multiple commits *as is*.
- if the *author* is a Pivotal employee, ensure that the author's email (i.e., in the `Author:` attribute of the commit) points to his or her actual `@gopivotal.com` address – for example, using `git commit --amend --author="Firstname Lastname <flastname@gopivotal.com>"`

Merge into master and push

Before merging to master, make sure that your local master fork is up to date.

```
$> git checkout master
$> git merge --no-ff --log -m "Merge pull request #<PULL_REQUEST_NUMBER> from <ACCOUNT>/<BRA
$> git push pivotal master:master
```

Generally speaking, `--no-ff` should be added for third-party contributions as it adds a separated commit for the merge. It shouldn't probably included when merging PRs from the team.

If you have merged the content of the branch and your local master fork was not up to date, you won't be able to push. You need to rebase your work in order to push

```
$> git fetch pivotal
$> git rebase --preserve-merges pivotal/master
```

Note also that the above commands assume that you have configured a `pivotal` remote similar to the following:

```
$> git remote show pivotal
* remote pivotal
  Fetch URL: git@github.com:spring-projects/spring-framework.git
  Push  URL: git@github.com:spring-projects/spring-framework.git
```

Backporting

Once you have merged a pull request into `master`, you should evaluate whether the change is a possible candidate for backporting. If so, create a `Backport` sub-task for the JIRA issue corresponding to the pull request and schedule it for the appropriate *Maintenance* version (e.g., "3.1 Maintenance"). This is the *'fire and forget'* model for backporting, in which someone else comes along later

and processes backports in bulk (at which point they are slated for a concrete maintenance release, e.g. 3.1.3).

See JIRA for example backport sub-tasks.

Cleaning Up

Update remote pull request branch to rebased/modified version (optional)

This is only possible if you have write permissions for the remote repository – for example, if you are merging your own pull request.

```
$> git push --force <ACCOUNT> <BRANCH>
```

Delete remote pull request branch (optional)

This is only possible if you have write permissions for the remote repository – for example, if you are merging your own pull request.

```
$> git push <ACCOUNT> :<BRANCH>
```

Delete local branch (optional)

```
$> git branch -D <BRANCH>
```

Delete local remote entry (optional)

```
$> git remote rm <ACCOUNT>
```