Git Branching Strategy

Matthew Kuperus Heun

May 20, 2024

Abstract

This document gives my strategy for branching in a git repository. The strategy is based on the fine blog post by Vincent Driessen and other resources. Adherence to this strategy will enable better development processes.

1 Introduction

The use of branches can greatly assist software development projects, particularly when developers need to both work on new features and maintain released code.

This document gives my model for branching in a git repository. The strategy is based on the fine blog post by Vincent Driessen [1] and other resources [2, 3]. The model assumes the existence of `github` or a similar git-based remote repository and use of the RStudio IDE, although the model (if not the details) will apply to other IDEs that support branching.

In addition, it can be difficult to remember all of the `git` commands for branching and merging. This document gathers all commands in a single place and discusses the commands in the context of the overall workflow.

Adherence to this strategy and use of these commands will improve my software development processes.

2 Rules

1. The `master/main` and `develop` branches are persistant
2. Only one (1) `release-x.x.x` branch can exist at any given time
3. Only one (1) `hotfix-x.x.x` branch can exist at any given time
4. Any number of feature branches can co-exist
5. Releases are retrieved by reverting to tags

3 Workflows

The workflows presented here assume familiarity with the Driessen branching model [1]. (See attached figure.) All `git` commands must be entered at the terminal of the developer's local computer in the root directory of the repository. Both Terminal.app (MacOS) and the Terminal window of RStudio can be used for this purpose.

3.1 Setup

Use the following workflow to establish a repository and create the `master/main` (release) and `develop` branches.

1. Create a git repository
2. Clone the repository, duplicating the `master/main` branch on the local computer: `git clone git@github.com:MatthewHeun/<<RepositoryName>>.git`
3. Create the `develop` branch: `git checkout -b develop master/main`
4. Make an initial push of the `develop` branch to origin (github): `git push -u origin develop`
5. Future pushes to github from the `develop` branch can be made from RStudio's `git interface`.
6. If needed, click the refresh icon ( ) in the Git pane of RStudio to show that the project is now on the `develop` branch.

To set up a repository with Zenodo for DOI creation, see `https://guides.github.com/activities/citable-code/`

To add a DOI badge to `Readme.md` or `Readme.Rmd`, see `https://gist.github.com/seignovert/ae6771f400ca464d294261f42900823a`

3.2 Develop a feature (`newfeature`)

Use the following steps to develop a feature.

1. Switch to develop branch: `git checkout develop`
2. Create a feature branch: `git checkout -b newfeature develop`
3. Work on the feature
4. Add new features to `NEWS.md` file as they are developed, but don't add a version number or date yet.
5. Commit changes locally
6. If `newfeature` is to be pushed to github, RStudio won't be able to do so until you

(a) Make an initial push of the `newfeature` branch to the origin (github): `git push -u origin newfeature`

(b) Future pushes to github can be made using the RStudio interface.

7. When complete, merge `newfeature` with the `develop` branch:

   (a) Switch to develop branch: `git checkout develop`

   (b) Merge `newfeature` into develop: `git merge --no-ff newfeature`

   (c) Add a commit comment (or accept the default) using `vi`

   (d) Save the comment using `ZZ` ("save" in `vi`)

8. If needed, click the refresh icon ( ) to show that the changes have been merged properly.

9. Push changes on `develop` branch to github

10. When satisfied, delete `newfeature` branch

    (a) Switch to develop branch: `git checkout develop`

    (b) Delete locally: `git branch -d newfeature`

    (c) Delete remote (if necessary): `git push -d origin newfeature`

    (d) Prune from other machines (if necessary): `git remote prune origin`

11. If intending to submit to CRAN, run the following steps.

    (a) `devtools::spell_check()`

    (b) `devtools::check_win_release()`

    (c) `devtools::check_win_devel()`

    (d) `devtools::check_win_oldrelease()`

    (e) `devtools::check_rhub(interactive = FALSE)`

    (f) `revdepcheck::revdep_reset()`

    (g) `revdepcheck::revdep_check(num_workers = 4)`

        i. Note that `revdepcheck` must be installed from github. See `https://github.com/r-lib/revdepcheck`.

        ii. `devtools::install_github("r-lib/revdepcheck")`

    (h) `devtools::release()`
    But don't actually release to CRAN!

3.3 Prepare for release

Use the following steps to prepare for a release.

1. Switch to develop branch: `git checkout develop`
2. Create a release branch: `git checkout -b release-x.x.x develop`
3. If the `release-x.x.x` branch is to be pushed to github (necessary if you want to submit to CRAN), RStudio won't be able to do so until you
   (a) Commit locally
   (b) Make an initial push of the `release-x.x.x` branch to the origin (github): `git push -u origin release-x.x.x`
   (c) Future pushes to github can be made using the RStudio interface.
4. Update documentation to reflect new version (`x.x.x`) and date. At a minimum:
   (a) Update `DESCRIPTION`, including `Version:` and `Date:` fields
   (b) Update `NEWS.md` with version number, date, and release notes
   (c) Update `inst/CITATION` version number in `note` and `textVersion` fields.
   (d) Update `LICENSE.md` in a new year.
   (e) Update `cran-comments.md` if submitting to CRAN.
   (f) `cmd-shift-B` to build the package
   (g) `cmd-shift-T` to test the package
   (h) `devtools::test_coverage()` to compute test coverage
   (i) `cmd-shift-E` to check the package
   (j) `cmd-shift-W` to update the GitHub pages website with new version number and documentation. Alternatively, `pkgdown:::build_site_external()` at the console.
5. Commit locally
6. Optionally, push to GitHub

3.4 Release

Use the following steps to release a new version (`x.x.x`).

1. If submitting to CRAN:
   (a) `devtools::spell_check()`
   (b) `devtools::check_win_release()`

(c) `devtools::check_win_devel()`

(d) `devtools::check_win_oldrelease()`

(e) `devtools::check_rhub(interactive = FALSE)`

(f) `revdepcheck::revdep_reset()`

(g) `revdepcheck::revdep_check(num_workers = 4)`

i. Note that `revdepcheck` must be installed from github. See `https://github.com/r-lib/revdepcheck`.

ii. `devtools::install_github("r-lib/revdepcheck")`

(h) `devtools::release()`
If `ERROR: No upstream branch` occurs, push the `release-x.x.x` branch to GitHub using `git push -u origin release-x.x.x`

(i) Answer all questions

(j) Verify submission to CRAN (via email)

(k) Wait for email acknowledging package is on its way to CRAN

2. Merge `release-x.x.x` into `master/main`

(a) Switch to master/main branch: `git checkout master/main`

(b) Perform merge: `git merge --no-ff release-x.x.x`

(c) Add a commit comment (or accept the default) using `vi`

(d) Save the comment using `ZZ` ("save" in `vi`)

3. Push `master/main` to github

4. Tag the release

(a) Tag local: `git tag -a vx.x.x`

(b) Tag remote (github)

i. Navigate to github page for the repository (`http://github.com/MatthewHeun/<<RepositoryName>>`)

ii. Click the releases button (0 releases)

iii. Create a new release

(c) Tag admin (for reference):

i. List local tags: `git tag -l`.

ii. List remote tags: `git ls-remote --tags origin`.

iii. Pull remote tags to local repo: `git fetch --tags`.

iv. Check out at a tag: `git checkout tags/vx.x.x`.

5. Merge `release-x.x.x` into `develop` branch

(a) Switch to develop branch: `git checkout develop`

(b) Perform merge: `git merge --no-ff release-x.x.x`

(c) Add a commit comment (or accept the default) using `vi`

(d) Save the comment using `ZZ` ("save" in `vi`)

6. Fix any conflicts that arise in the `develop` branch
7. Push `develop` branch to github
8. When satisfied, delete `release-x.x.x` branch
   (a) Switch to develop branch: `git checkout develop`
   (b) Delete locally: `git branch -d release-x.x.x`
   (c) Delete remote (if necessary): `git push -d origin release-x.x.x`
   (d) Prune from other machines (if necessary): `git remote prune origin`
9. Add Zenodo badges
   (a) Go to the repository's page on Zenodo.
   (b) Copy the markdown text for the Zenodo badge.
   (c) Switch to `develop` branch.
   (d) Paste Zenodo badge text into `NEWS.md`.

3.5 Perform a hotfix

Execute the following steps to perform a hotfix to the current release.

1. Switch to master/main branch: `git checkout master/main`
2. Create hotfix branch: `git checkout -b hotfix-x.x.x master/main`
3. Develop the fix
4. Update documentation to reflect new version (`x.x.x`) and date. At a minimum:
   (a) Update `DESCRIPTION`, including `Version:` and `Date:` fields
   (b) Update `NEWS.md` with version number, date, and release notes
   (c) Update `inst/CITATION` version number in `note` and `textVersion` fields.
   (d) Update `LICENSE.md` in a new year.
   (e) Update `cran-comments.md` if submitting to CRAN.
   (f) Re-run `cmd-shift-W` or `Build pkgdown` from Addins menu or `pkgdown:::build_site_external()` at the console. This step updates the version number on `pkgdown` website documentation.
5. Prepare release notes
6. Commit locally

7. If `hotfix-x.x.x` is to be pushed to github, RStudio won't be able to do so until you
   (a) Make an initial push of the `hotfix-x.x.x` branch to the origin (github): `git push -u origin hotfix-x.x.x`
   (b) If needed, click the refresh icon ( ) in the Git pane of RStudio to show that the branch can be pushed to origin (github)
   (c) Future pushes to github can be made using the RStudio interface.
8. Merge `hotfix-x.x.x` into `master/main`
   (a) Switch to master/main branch: `git checkout master/main`
   (b) Perform merge: `git merge --no-ff hotfix-x.x.x`
   (c) Add a commit comment (or accept the default) using `vi`
   (d) Save the comment using `ZZ` ("save" in `vi`)
   (e) Push `master/main` branch to GitHub
9. Tag the release
   (a) Tag local: `git tag -a vx.x.x`
   (b) Tag remote (github)
      i. Navigate to github page for the repository
      ii. Click the releases button (0 releases)
      iii. Create a new release
10. *If a release branch exists*, merge `hotfix-x.x.x` into the release branch
    (a) Switch to the release branch: `git checkout release-x.x.x`
    (b) Attempt merge: `git merge --no-ff hotfix-x.x.x`
    (c) Fix conflicts as needed
    (d) When successful, add a commit comment (or accept the default) using `vi`
    (e) Save the comment using `ZZ` ("save" in `vi`)
    (f) Push the release branch to GitHub, if necessary
11. *If no release branch exists*, merge `hotfix-x.x.x` into `develop`
    (a) Switch to `develop branch`: `git checkout develop`
    (b) Perform merge: `git merge --no-ff hotfix-x.x.x`
    (c) Fix conflicts as needed
    (d) Add a commit comment (or accept the default) using `vi`
    (e) Save the comment using `ZZ` ("save" in `vi`)
    (f) Push the `develop` branch to GitHub

12. When satisfied, delete `hotfix-x.x.x` branch
    (a) Switch to `master/main` branch: `git checkout master/main`
    (b) Delete locally: `git branch -d hotfix-x.x.x`
    (c) Delete remote (if necessary): `git push -d origin hotfix-x.x.x`
    (d) Prune from other machines (if necessary): `git remote prune origin`

4 Conclusion

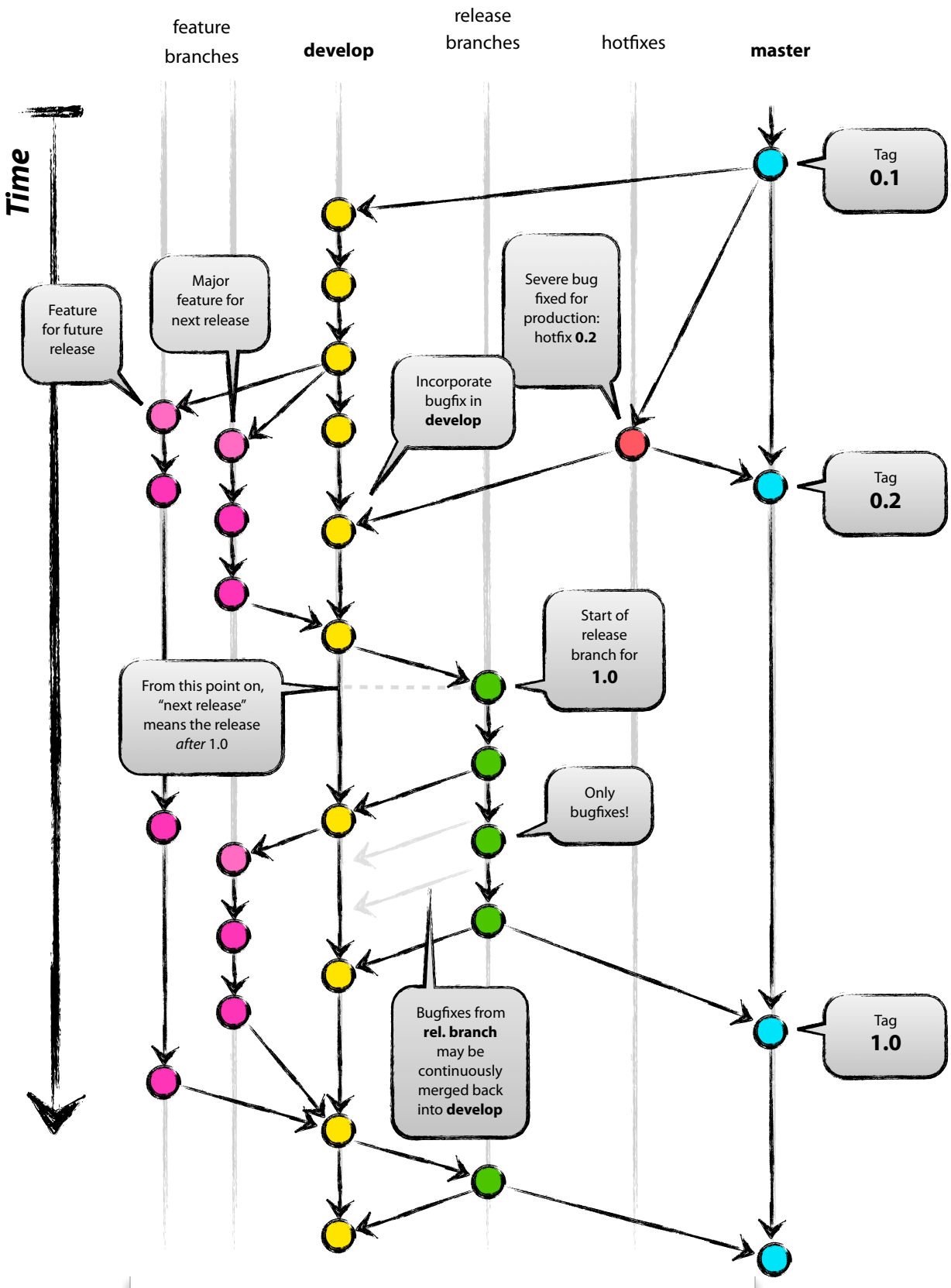
The rules and workflows above will greatly assist development of `R` packages.

References

[1] V. Driessen. A successful git branching model. `http://nvie.com/posts/a-successful-git-branching-model/`, 5 January 2010.

[2] T. Onkelinx. Response to "push branch" question from Ian Pylvainen. `https://support.rstudio.com/hc/en-us/community/posts/200666727-push-branch`, May 2017.

[3] M. Rankin. Response to "Failed attempts to delete remote branch" by Matthew Rankin. `https://stackoverflow.com/questions/2003505/how-do-i-delete-a-git-branch-both-locally-and-remotely#2003515`, 5 January 2010.



Author: Vincent Driessen
Original blog post: http://nvie.com/posts/a-succesful-git-branching-model