Update R development process

R.G. Thomas

2025-02-02

Table of contents

1 Introduction **2**

2 Making Changes to an R Package: A Complete Workflow **5**

2.1 Initial Code Changes 5
2.2 Local Testing and Building 6
2.3 GitHub Integration 6
2.4 GitHub Actions 7
2.4.1 R-CMD-check 7
2.4.2 Test Coverage 7
2.4.3 pkgdown 7
2.5 Setting Up GitHub Actions 7
2.6 Monitoring the Process 8
2.7 Best Practices 8
2.8 Conclusion 8
2.9 Further Reading and Resources 9
2.9.1 Git Fundamentals 9
2.9.2 R Package Development 9
2.9.3 GitHub Actions 9
2.9.4 Testing in R 9
2.9.5 Package Documentation 10
2.9.6 Interactive Learning 10

3 Git Fundamentals **10**

4 R Package Development **10**

5 GitHub Actions **11**

6 Testing in R **11**

7 Package Documentation **11**

8 Interactive Learning **11**

References **11**

1 Introduction

Figure 1: quarto

Here's the expanded procedure including Git and GitHub steps:

1. Create New Branch

```
git checkout main          # Start from main branch
git pull origin main      # Get latest changes
git checkout -b feature-name   # Create & switch to new branch
```

2. Edit the R Code

   - Navigate to R/zzdataframe2graphic.R
   - Make your code changes
   - Document new parameters/changes in roxygen comments
   - Save file

3. Update Documentation

```
# In R:
devtools::document()   # Updates NAMESPACE and man/ files
```

4. Write New Tests

   - Edit tests/testthat/test-zzdataframe2graphic.R
   - Add new test cases:

```
test_that("new feature works as expected", {
  result <- your_function(test_input)
  expect_equal(result, expected_output)
})
```

5. Run Tests Locally

```
devtools::test()
# Fix any failures and add more tests if needed
```

6. Update Version & Description

   - Edit DESCRIPTION file
   - Increment version number
   - Update date field
   - Update any dependencies if needed

7. Build and Check Package

```
devtools::build()
devtools::check()
```

8. Additional Checks

```
devtools::check_win_devel()
devtools::check_mac_release()
rcmdcheck::rcmdcheck(args = c("--as-cran"))
```

9. Stage Changes

```
git status                    # Review changes
git add R/zzdataframe2graphic.R
git add tests/testthat/test-zzdataframe2graphic.R
git add DESCRIPTION
git add man/*
git add NAMESPACE
```

10. Commit Changes

```
git commit -m "feat: add new feature to zzdataframe2graphic

- Added X functionality
- Updated documentation
- Added tests"
```

11. Push to GitHub `bash` `git push origin feature-name`
12. Create Pull Request
    - Go to your GitHub repository
    - Click “Pull requests” tab
    - Click “New pull request”
    - Select:
        - base: main
        - compare: feature-name
    - Fill in PR description:
        - What changes were made
        - Why changes were made
        - How to test the changes
        - Any related issues
13. CI/Actions Check
    - Wait for GitHub Actions to complete
    - Review any failures
    - Make necessary fixes:

```
git add .
git commit -m "fix: address CI failures"
git push origin feature-name
```

14. PR Review Process
    - Address any review comments
    - Make requested changes
    - Push additional commits
    - Request re-review if needed
15. Merge PR
    - Once approved and all checks pass
    - Choose merge strategy (usually “Squash and merge”)
    - Update PR title if needed
    - Click “Squash and merge”

16. Clean Up `bash git checkout main git pull origin main git branch -d feature-name # Delete local branch`
17. Create Release (optional)
    - Go to GitHub "Releases"
    - Click "Create new release"
    - Tag version (e.g., v0.2.1)
    - Add release notes
    - Publish release

Would you like me to elaborate on any of these steps or explain the reasoning behind any particular practices? Here's an expanded blog post about making changes to the zzdataframe2graphic package:

2 Making Changes to an R Package: A Complete Workflow

In this post, I'll walk through the complete process of making changes to an R package, from code modification to testing and GitHub integration. I'll use my package `zzdataframe2graphic` as an example.

2.1 Initial Code Changes

The workflow begins with modifying your R code. Let's say we want to improve error handling in the package:

1. Navigate to the package's R directory
2. Open the relevant R file (in this case, `zzdataframe2graphic.R`)
3. Make your changes
4. Document any new parameters or changes in the roxygen comments

2.2 Local Testing and Building

After making changes, follow these steps:

1. Document the changes:

```
devtools::document()
```

This updates the NAMESPACE and man/ files based on your roxygen comments.

2. Run the tests:

```
devtools::test()
```

Fix any test failures that appear.

3. Build and install the package:

```
devtools::build()
devtools::install()
```

2.3 GitHub Integration

Once your local changes are working, it's time to integrate with GitHub:

1. Create a new branch:

```
git checkout -b feature-name
```

2. Stage and commit your changes:

```
git add R/zzdataframe2graphic.R
git add tests/testthat/test-zzdataframe2graphic.R
git commit -m "Add improved error handling"
```

3. Push to GitHub:

```
git push origin feature-name
```

2.4 GitHub Actions

GitHub Actions automate the testing process. The package uses three main workflows:

2.4.1 R-CMD-check

This workflow runs `R CMD check` on multiple operating systems: - Windows latest - macOS latest - Ubuntu latest (with R-release, R-devel, and R-oldrel)

The workflow is triggered on: - Push to main/master - Pull requests to main/master

2.4.2 Test Coverage

This workflow: - Runs all package tests - Generates coverage reports - Requires proper setup of LaTeX dependencies - Reports which parts of the code need more testing

2.4.3 pkgdown

This workflow: - Builds package documentation - Deploys to GitHub Pages - Updates whenever changes are pushed to main/master

2.5 Setting Up GitHub Actions

To set up these workflows:

1. Create the workflows directory:

```
mkdir -p .github/workflows
```

2. Add the necessary YAML files:

- `R-CMD-check.yaml`
- `test-coverage.yaml`
- `pkgdown.yaml`

3. Configure repository settings:

- Go to Settings → Actions → General
- Set "Workflow permissions" to "Read and write"
- Enable GitHub Pages deployment

2.6 Monitoring the Process

After pushing changes:

1. Check the "Actions" tab on GitHub
2. Monitor each workflow's progress
3. Review any failures
4. Check the generated documentation site
5. Review test coverage reports

2.7 Best Practices

- Always create a new branch for changes
- Run tests locally before pushing
- Write tests for new functionality
- Keep commits focused and well-documented
- Monitor GitHub Actions results
- Address failures promptly

2.8 Conclusion

This workflow ensures that changes to the package are properly tested, documented, and integrated. The GitHub Actions automation provides confidence that the package works across different environments and maintains high-quality documentation.

Would you like me to expand on any particular aspect of this workflow?

Here's an additional section for the blog post with references:

2.9 Further Reading and Resources

2.9.1 Git Fundamentals

- Pro Git Book - The official and comprehensive guide to Git
- Git Handbook - GitHub's introduction to Git basics
- Oh Shit, Git!?! - Practical solutions for common Git mistakes

2.9.2 R Package Development

- R Packages (2e) by Hadley Wickham and Jenny Bryan
- Writing R Extensions - The official R manual for package development
- devtools documentation - Comprehensive guide to the devtools package

2.9.3 GitHub Actions

- GitHub Actions for R - r-lib's collection of GitHub Actions for R
- GitHub Actions Documentation - Official documentation
- GitHub Actions for R packages - Examples from the usethis package

2.9.4 Testing in R

- testthat documentation - The standard testing framework for R
- Testing R packages - Chapter from R Packages book
- covr documentation - For understanding test coverage

2.9.5 Package Documentation

- roxygen2 documentation - For writing package documentation
- pkgdown documentation - For creating package websites
- Documentation guide from the tidyverse style guide

2.9.6 Interactive Learning

- GitHub Skills - Interactive courses for learning GitHub
- Try Git - Resources to learn Git
- GitHub Learning Lab - Interactive GitHub tutorials

These resources range from beginner-friendly introductions to detailed technical references, allowing readers to dive deeper into any aspect of the workflow that interests them.

3 Git Fundamentals

The official and comprehensive guide to Git is available in the Pro Git Book Chacon and Straub (2014). GitHub provides an excellent introduction through their Git Handbook "Git Handbook" (2024). For practical solutions to common Git problems, the Oh Shit, Git!?! guide Sylor-Miller (2024) is invaluable.

4 R Package Development

The definitive guide is "R Packages (2e)" by Hadley Wickham and Jenny Bryan Wickham and Bryan (2023). The official manual for package development is "Writing R Extensions" *Writing r Extensions* (2024). For practical development tools, see the devtools documentation Wickham et al. (2024a).

5 GitHub Actions

The r-lib collection of GitHub Actions for R "GitHub Actions for r" (2024) provides essential tools. The official GitHub Actions documentation "GitHub Actions Documentation" (2024) offers comprehensive coverage. The usethis package provides practical examples "GitHub Actions for r Packages" (2024).

6 Testing in R

The standard testing framework is documented in testthat Wickham (2024a). For comprehensive coverage of testing, see the testing chapter in R Packages Wickham and Bryan (2023). Understanding test coverage is explained in the covr documentation Hester (2024).

7 Package Documentation

Package documentation tools include roxygen2 Wickham et al. (2024c) and pkgdown Wickham et al. (2024b). The tidyverse style guide provides documentation standards Wickham (2024b).

8 Interactive Learning

GitHub provides interactive learning through GitHub Skills "GitHub Skills" (2024) and GitHub Learning Lab "GitHub Learning Lab" (2024). The Try Git resources "Try Git" (2024) offer hands-on practice.

References

Chacon, Scott, and Ben Straub. 2014. *Pro Git*. 2nd ed. Apress. https://git-scm.com/book/en/v2.

"Git Handbook." 2024. GitHub. 2024. https://guides.github.com/introduction/git-handbook/.
"GitHub Actions Documentation." 2024. GitHub. 2024. https://docs.github.com/en/actions.
"GitHub Actions for r." 2024. r-lib. 2024. https://github.com/r-lib/actions.
"GitHub Actions for r Packages." 2024. usethis. 2024. https://usethis.r-lib.org/articles/articles/usethis-github-action-examples.html.
"GitHub Learning Lab." 2024. GitHub. 2024. https://lab.github.com/.
"GitHub Skills." 2024. GitHub. 2024. https://skills.github.com/.
Hester, Jim. 2024. "Covr: Test Coverage for r Packages." 2024. https://covr.r-lib.org/.
Sylor-Miller, Katie. 2024. "Oh Shit, Git!?!" 2024. https://ohshitgit.com/.
"Try Git." 2024. GitHub. 2024. https://try.github.io/.
Wickham, Hadley et al. 2024a. "Devtools: Tools to Make Developing r Packages Easier." 2024. https://devtools.r-lib.org/.
——— et al. 2024b. "Pkgdown: Make Static HTML Documentation for an r Package." 2024. https://pkgdown.r-lib.org/.
——— et al. 2024c. "Roxygen2: In-Line Documentation for r." 2024. https://roxygen2.r-lib.org/.
Wickham, Hadley. 2024a. "Testthat: Unit Testing for r." 2024. https://testthat.r-lib.org/.
———. 2024b. "The Tidyverse Style Guide." 2024. https://style.tidyverse.org/documentation.html.
Wickham, Hadley, and Jenny Bryan. 2023. *R Packages (2nd Edition)*. O'Reilly. https://r-pkgs.org/.
Writing r Extensions. 2024. R Core Team. https://cran.r-project.org/doc/manuals/r-release/R-exts.html.