There’s one thing that has bugged me for a while about developing R packages. We have all these nice, modern tools we have for tracking our code, producing web sites from the **roxygen** documentation, an so on. Yet for every code commit I make to the master branch of a package repo, there’s often two or more additional steps I need to take to keep the package README.md and *pkgdown* site in sync with the code. Don’t get me wrong; it’s amazing that we have these tools available to help users get to grips with our R packages. It’s just that there’s a lot of extra things to remember to do to keep everything up to date. The development of free-to-use services such as Travis CI or Appveyor have been very useful as they can automate many of these repetitive tasks. The other day I was grappling with getting a GitHub Actions workflow to render a README.Rmd file to README.md on GitHub, so that I didn’t have to do it locally all the time. After a lot of trial and error, this is how I got it working.

The general use case I am imagining here is the package author that has a README.Rmd file that contains R code chunks, which they want to render to README.md so it will get displayed nicely on GitHub. You might want to do this to provide a simple overview of how to use some key functionality of your package or show off a plot or two that can be generated by the package. It’s pretty easy to render this locally with a Makefile or by simply invoking the correct R incantation directly in the terminal. However, wouldn’t it be great if we could automate this!

The first step in getting this working was to recognise that the R Infrastructure organisation has been working to make R-related GitHub Actions workflows available to users. This effort has been lead by Jim Hester and Jim has very helpfully provided a workflow example YAML file showing how one might go about rendering a README.Rmd file to README.md using the **rmarkdown** package.

Also, the **usethis** package has made it incredibly easy to get started using GitHub Actions; **usethis** provides use\_github\_actions() to set your package up to start using GitHub Actions to check your package builds without errors. There’s also a use\_github-action() function that can add workflows from the r-lib/actions repo to your package.

If you don’t have **usethis** installed, install it (install.packages(“usethis”)), then you can set your R package repo up to run R CMD check on your package on GitHub’s servers by running

usethis::use\_github\_actions()

in an R session in the package root folder. Running this will produce something like this

> usethis::use\_github\_actions()

✔ Setting active project to '/home/gavin/work/git/gratia/gratia'

✔ Creating '.github/'

✔ Adding '\*.html' to '.github/.gitignore'

✔ Creating '.github/workflows/'

✔ Writing '.github/workflows/R-CMD-check.yaml'

● Copy and paste the following lines into '/home/gavin/work/git/gratia/gratia/README.md':

which outlines the steps **usethis** has taken on your behalf. The last line prints out some text that you can paste into the README.Rmd to show a status badge for the GitHub Action; in this case it will show whether or not your package passed R CMD check without error.

This also nicely illustrates how you might set things up by hand of course, especially if you don’t want to run R CMD check on each push.

GitHub Actions workflows are configurations that describe the steps in the workflow and are stored in YAML files. These files should be located in a .github/workflows folder in the package root. If all you want to do is render a README.Rmd to README.md you could just as easily create this folder yourself. I’m not sure why **usethis** also creates a .gitignore containing ’\*.html’ in the .github folder, but if this is needed for what you’re doing, go ahead and create it too.

To get set-up quickly to render README.Rmd to markdown, you can now use use\_github\_action(“render-readme.yaml”). This will copy the render-readme.yaml file from [r-lib/actions/examples](https://github.com/r-lib/actions/tree/master/examples) to .github/workflows/render-readme.yaml. Alternatively, you can touch .github/workflows/render-readme.yaml and add what you need by hand.

This is what the contents of render-readme.yaml look like, at the time of writing, if you used **usethis** to create it:

on:

push:

paths:

- README.Rmd

name: Render README

jobs:

render:

name: Render README

runs-on: macOS-latest

steps:

- uses: actions/checkout@v2

- uses: r-lib/actions/setup-r@v1

- uses: r-lib/actions/setup-pandoc@v1

- name: Install rmarkdown

run: Rscript -e 'install.packages("rmarkdown")'

- name: Render README

run: Rscript -e 'rmarkdown::render("README.Rmd")'

- name: Commit results

run: |

git commit README.md -m 'Re-build README.Rmd' || echo "No changes to commit"

git push origin || echo "No changes to commit"

The first bit under on: controls when the workflow is triggered. The way the example workflow is set up means it will only be triggered *if* a file matching the path README.Rmd is included in the commit when pushed to the repo. It’s also worth noting that *until the workflow is actually triggered*, it won’t show up in the *Actions* tab in your repo on GitHub — this caused me no end of grief until I figured our this GitHub Actions *feature*. To trigger this workflow, you need to edit README.Rmd, add and commit those changes using git, and then push the changes to GitHub.

That didn’t suit my use case however; what if I change the package code in such a way that any output or plots produced by code in the README.Rmd would also change? In this case, I would have to needlessly tweak something in README.Rmd and push that change just to trigger rendering.

There’s probably a better way to do this — such as setting paths: to a wildcard that would match *any* .R file in the R folder so the workflow would be triggered on any change to the package code — but to just get something up and running I changed the on: part to read:

on:

push:

branches: master

which indicates that the workflow should run for any push to the *master* branch of the repo.

The top-level name: element is how your workflow will be listed in the Actions tab in your repo. Set this to something short but descriptive so it is easy to filter the various outputs from workflows that are run on the GitHub Actions service.

All workflows contain one or more *jobs*, listed under the jobs: element. In the example YAML file, there is a single job listed as render:, which has a name, Render README.

The runs-on element indicates what system the job will be run on; here is is a Mac OS system. I’m not sure why the *r-lib/actions* example workflows all run on Mac OS systems? Anyway, they work, so no need to change that unless you need something specific.

The steps: section is where the stages of the job are defined.

steps:

- uses: actions/checkout@v2

- uses: r-lib/actions/setup-r@v1

- uses: r-lib/actions/setup-pandoc@v1

Each of the uses: elements pulls in some pre-existing worfklow steps that you can build to upon to bootstrap the solution you need. For example, the actions/checkout@v2 workflow contains everything you need to checkout your repo and make it available to the current job. This is pretty fundamental; unless the GitHub Actions service can get at the code in your repo, it won’t be able to do anything useful whatsoever.

The next two uses: are workflows provide by *r-lib/actions* that set up a working R installation (r-lib/actions/setup-r@v1) and the **Pandoc** library used by **rmarkdown** (r-lib/actions/setup-pandoc@v1).

After the uses: declarations, the YAML file includes a series of steps that describe commands that are run on the service. This is where the real action takes place.

- name: Install rmarkdown

run: Rscript -e 'install.packages("rmarkdown")'

- name: Render README

run: Rscript -e 'rmarkdown::render("README.Rmd")'

- name: Commit results

run: |

git commit README.md -m 'Re-build README.Rmd' || echo "No changes to commit"

git push origin || echo "No changes to commit"

Here we see three sets of commands that will be run

1. the first installs the **rmarkdown** package,
2. the second runs rmarkdown::render() on README.Rmd to render it, and
3. the third commits the rendered README.md file and pushes it to your repo, or echos a comment if no changes are needed.

Notice how the run: element for the last step has a | after run:. This indicates that this particular step involves multiple lines of commands to be executed one after another.

If you’ve not come across Rscript before, it’s a way to use R like a scripting language, non-interactively. Here we’re using the -e flag to tell Rscript what R code to run, rather than passing it a .R to run.

Out of the box, these steps aren’t going to be very useful for R package maintainers if the README.Rmd uses anything other than the base R installation and recommended packages. At the very least you are going to want to also install the R package you are documenting in the README.Rmd, plus any other packages you need for the Rmd that might not be dependencies of the package in the repo.

In my case, I just needed to install the **gratia** package alongside **rmarkdown**, so I changed that run: element to be

- name: Install rmarkdown

run: Rscript -e 'install.packages(c("rmarkdown", "gratia"))'

I also decided to change the rmarkdown::render() call; by default this will generate HTML output by rendering the .Rmd first to .md and thence to .html. As we don’t need this latter step, I changed the output\_format argument of render() to be “md\_document”, so that element now looks like this

- name: Render README

run: Rscript -e 'rmarkdown::render("README.Rmd", output\_format = "md\_document")'

Doing this means I don’t also generate a README.html file (which might be why the .gitignore was created by **usethis** earlier?); keeping the .gitignore can’t hurt given that it only excludes any .html files from a commit, so I left it alone.

I also modified the *commit* step too. The default assumes you already have a README.md in the repo and that this is the only file you want to add to the commit. If you render any plots in the .Rmd, then you’ll also want to add those to the commit. So, I added an explicit git add line prior to the commit, and also simplified the latter

- name: Commit results

run: |

git add README.md man/figures/README-\*

git commit -m 'Re-build README.Rmd' || echo "No changes to commit"

git push origin || echo "No changes to commit"

As you can see, I used a wildcard to catch any figures created by the render. In the README.Rmd I used a setup chunk to set the fig.path **knitr** option so that any plots were generated in the man/figures folder and had the prefix README- prepended to the file name:

knitr::opts\_chunk$set(

fig.path = "man/figures/README-"

)

The man/figures folder is a useful place to store figures generated like this as they’ll be carried along with your R package and available on CRAN, where the README.md file is also displayed if present. This folder is also used if you generate and include figures in the package documentation using **roxygen**, for example.

I used the prefix README- so that I could limit what I was adding in the git add step of the workflow. I’m always a bit nervous when staging files for a commit and never use git commit -a for example. This way I have a reasonable means of only adding plots that were created by rendering README.Rmd.

After these changes (and a few others as I was troubleshooting some issues) my workflow to render README.Rmd files looks like this

name: render readme

# Controls when the action will run

on:

push:

branches: master

jobs:

render:

# The type of runner that the job will run on

runs-on: macOS-latest

steps:

# Checks-out your repository under $GITHUB\_WORKSPACE, so your job can access it

- uses: actions/checkout@v2

- uses: r-lib/actions/setup-r@v1

- uses: r-lib/actions/setup-pandoc@v1

# install packages needed

- name: install required packages

run: Rscript -e 'install.packages(c("rmarkdown","gratia"))'

# Render READEME.md using rmarkdown

- name: render README

run: Rscript -e 'rmarkdown::render("README.Rmd", output\_format = "md\_document")'

- name: commit rendered README

run: |

git add README.md man/figures/README-\*

git commit -m "Re-build README.md" || echo "No changes to commit"

git push origin master || echo "No changes to commit"

This is a first pass at getting something working; it’s just occurred to me that the git add line probably needs to be linked with the git commit line so it only tries to commit if files were staged with git add.

It would also be good to try to cache the installed packages so the workflow doesn’t need to install everything for **rmarkdown** and **gratia** every time it is run. However, I was running into issues related to the R 4.0.0 release and packages in the cache not getting refreshed even though they were out of date. So I removed that step from my pkgdown.yaml workflow, and as a result didn’t try to implement it for rendering README.Rmd files. Yet anyway…

For reference the workflow takes between two and three minutes to run on GitHub, even without package caching, which isn’t too bad, but rendering the README.Rmd locally takes only a few seconds, so there’s lots to be gained here by figuring out a reliable caching mechanism.

If you have implemented something similar for a GitHub Actions workflow, let me know in the comments below; this is all quite new to me and I’m interested in how other people might have tackled this. Now that I have this working reliably I only need to remember to git pull from GitHub more often to get the changes to README.md. The next issue I want to look at is getting the right paths: settings so the README.Rmd is rendered only when relevant files are changed in the package, not on every push to the repo.

Lastly, a big **thank you** to Jim Hester and everyone else who’s contributed to the R-related GitHub Actions workflows. This is an amazingly useful service for the R Community, and I for one am incredibly thankful that we have such helpful and knowledgeable people among us that are doing all this great work to make developing R packages that much easier.