output:
html_document: default

pdf_document: default

Task 3: How to integrate Git with R Studio

This task is designed for students and researchers who want to implement a system of version control within a standard R-based workflow. This can be applied to a range of software development, data analysis and project management tasks. Your future research self will thank your for the convenience.

Don't forget you can join in the discussions over at our open **Slack channel**. Please do introduce yourself at #module5opensource, and tell us a bit about who you are, your background, and how you ended up here!

Estimated time to complete: 30 minutes

Estimate time saving once complete: Virtually infinite

Table of contents

- Getting started
  - Git
  - R Studio
- Step one: Download all the things
- Step two: Configure Git inside RStudio
- Step three: Why did I just do that?
- Step four: The perfect marriage between Git and R
- Step five: Getting content with content
- Step six: A brave commitment
- Step seven: PUSH!

Getting started

Congratulations on making it this far! If you're reading this, you've survived pull requests, web-hooks, and can probably even tell us know what the F in FOSS stands for (*not* Frustration…) Hopefully, you have overcome any scepticism or reluctance towards the benefits of GitHub and Open Source Software, and are ready to take the next step.

Before starting this Task, please make sure you have already completed Task 1 and Task 2, so that you are more familiar with GitHub and some standard Open Source practices.

This task will teach you how to integrate the version control software, Git, with the popular coding environment, RStudio. And yes, it is Git as in gif or God, not Jit as in the wrong way of pronouncing things.

If you are one of those researchers who thinks that having code spread across multiple hard-drives that are waiting to break, Dropbox, Google Drive, or any other non-specialist software, this task is just for you. If you have ever experienced the mind-numbing process of having multiple 'final' versions of a paper bouncing between different co-authors, this is also for you.

All of us are guilty of these sorts of things once in a while, but there are ways to do it that are better for you, future you, and those who might benefit from your work.

Getting Git

So, what is Git, and how is it different to GitHub? Git is a version control system, which enables you to save and track time-stamped copies of your work throughout the development process. It also works with non-code items too, like this MOOC, the majority of which was written in markdown in RStudio, and integrated with a Git/GitHub workflow.

This is important, as all research goes through changes and sometimes we want to know what those things were. Did you delete some text that you now think is important? Version control will save that for you. Did your code used to work perfectly, but is now buggy beyond belief? Version control. It's a great way to avoid that chaotic state where you have multiple copies of the same file, but without a stupid and annoying file naming convention. `FINAL_Revised_2.2_supervisor_edits_ver1.7_scream.txt` will be a thing of the past.

GitHub is the platform that allows you to seamlessly share code from your workspace (e.g., laptop) to be hosted in an online space. So, sort of like the public interface to GitHub. The advantages of Git/GitHub are:

1. You get to keep copies of all your w ork through time;
2. You can compare w ork through different copies through time, w hich helps to spot bugs or errors;
3. Other people can collaborate openly w ith your w ork;
4. You have both a local and an online copy of your w ork that remain in sync;
5. It is fully transparent as to w ho made a contribution, w hy they made it, and w hen; and
6. You can have multiple people w orking on the same project at once in parallel.

While this w as primarily designed for source code, it should be instantly obvious how this becomes a pow erful tool for virtually all research w orkflow s.

RStudio

RStudio is a popular coding environment for researchers w ho use the statistical programming language, R. It comes w ith a text editor, so you don't have to install another and sw itch betw een. It also includes a graphical user interface (GUI) to Git and GitHub, w hich w e w ill be using here.

Isn't it nice w hen brilliant Open Source tools integrate seamlessly like that. This should help to make your daily use of Git much simpler.

If at any point you need to install new packages for R, simply use the follow ing command:

`install.packages("PACKAGE NAME", dependencies = TRUE)`

Replacing `PACKAGE NAME` w ith the, er, package name. Some examples you can play w ith that might come in useful include `knitr` , `devtools` or `ggplot2` .

Step one: Download all the things

1. You should already have a GitHub account by now if you have follow ed the previous tasks. If not, sign up here. Free unlimited repositories for all!
2. Dow nload and install the latest version of R. Also available for Mac and Linux.
3. Dow nload and install the latest version of Rstudio. Oh, hey, looks it Open Source! Sw ish.
4. Dow nload and install the latest version of Git. **Make sure to Select “Use Git from the Windows Command Prompt” during installation.**

> **Pro-tip**: To update all of your R packages in one, simply execute the follow ing code `update.packages(ask = FALSE, checkBuilt = TRUE)`

For now, just choose all the usual default options for each install. Depending on w hich Operating System (e.g., Mac, Window s, Linux), this might be different for each of you. For now, and for the rest of this task, w e're going to stick w ith doing things the easy-ish Window s w ay (but also provide some instructions for using the command line).

For Linux or Debian users, simply use the follow ing command to install Git:

`sudo apt-get install git-core`

For Mac users, this link, or purchase a new laptop w ith a different operating system.

If you w ant, you can also dow nload the local version of GitHub and use it through the simple GUI. It's available on Window s and Mac and Linux, and can make your life a little easier, especially if you w ant to use a different platform to RStudio.

> **Pro-tip:** You see w hen installing Git it says 'Use Git Bash as shell for Git projects?' This is the place w here you can use the command-line to access Git from outside of RStudio. It's a pow erful beast. Try the follow ing tw o commands to get started:

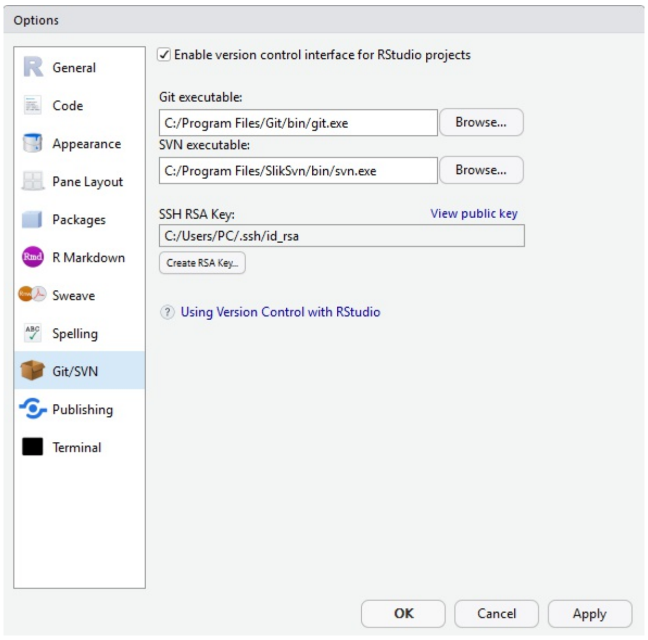
`git config --global user.name 'YOUR USERNAME'`

`git config --global user.email 'YOUR EMAIL'`

Step two: Configure Git inside RStudio

Right, that's the easy bit done. Next, go into RStudio, and in the tabs at the top go to Go to **Tools > Global Options > Git/SVN**. SVN is just another version control system like Git, and w e don't need to w orry about that here.

In the place w here it says *Git executable*, add the pathw ay here to the git.exe file that you just dow nloaded in the previous step. Make sure the boxe here that says **Enable version control interface for RStudio projects** is ticked. This now has tied version control to future projects in RStudio, to provide a really pow erful additional dimension to collaborative or solo w ork.



☑ Enable version control interface for RStudio projects

Git executable:
C:/Program Files/Git/bin/git.exe

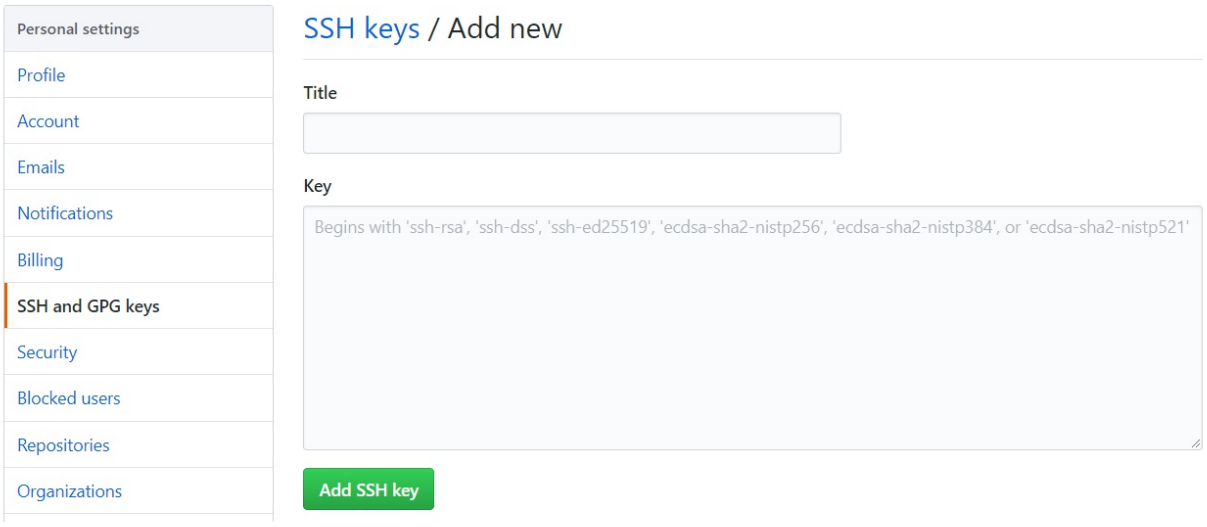
SVN executable:
C:/Program Files/SlikSvn/bin/svn.exe

SSH RSA Key:
C:/Users/PC/.ssh/id_rsa

The Global Options window inside RStudio

Next, hit the button in this w indow that says *Create RSA Key*, This is a private key that is used for authentication betw een different systems, and saves you from having to type in your passw ord over and over. Here, it w ill pop up a new w indow w ith a public key, that you w ant to copy to your clipboard.

Head over to GitHub, go to your profile settings, and the *SSH and GPG keys* tab. Click *New SSH key*. Here, paste in the key from RStudio, and call it something imaginative like 'RStudio'.



Inside GitHub where you will want to enter the key you just generated in RStudio

OK, now hold on to your butts, w e're going into the command line. Don't w orry if you've never used the shell before because it's quite similar to using R, or any other coding system. The main difference here though is that instead of calling functions like in R, you call commands.

So back in RStudio, go to **Tools > Shell**, and it w ill open up a command prompt w indow. If you already played w ith the Git Bash above, you should have done this step already. Enter the follow ing tw o commands:

```
git config --global user.name 'YOUR USERNAME'
```

```
git config --global user.email 'YOUR EMAIL'
```

Hopefully it does not have to be said to substitute in your own GitHub username and email here. You can access this at any point just by finding the 'Shell' within Windows. Or, if you right click on any folder on your Desktop that is linked to a GitHub repo, you can open up the Shell instantly and Bash away.

What this stage has done is configure Git, which is software that runs on your desktop, to GitHub, which is a repository website.

Restart R Studio. Whew, that was tough. Next.

Step three: Why did I just do that?

OK, hold your breathe, we're going to pause here just to learn some basic Git commands. Some of the key ones you could do with learning are:

- **Add**: This is where you submit files to the staging area before being committed.

- **Commit** This is like 'saving' your work by creating a new version or copy.

- **Push**: This is how you send files from your local project to the online repository.

- **Pull**: This is how you get files from your online repository to your local project.

Back in RStudio, type in the following into the *Terminal*, or by opening up a new Shell:

`git add .`

It won't actually do anything for now, but in the future will add all files in your current working directory (that's what the `.` does) to staging ready for a commit.

Step four: The perfect marriage between Git and R

Now, in Task 1, you should have learned how to build your very first GitHub repository. If you haven't done that, we can wait here while you go and do that. If you have already, or have an existing GitHub repository, we can move on.

So, you should have a repository on GitHub, complete with a `README` file, a `LICENSE` file and some other bits and bobs.

What we are going to do now, is integrate that repository with Git. Steady now.

1. Firstly, go to **Project > Create Project > Version Control > Git**.
2. Back on GitHub, you should see a bit where there is a https:// URL. That is the link to your repository, and it gives you the option to clone it in your desktop. For now, just copy that link, switch back to RStudio, and paste it into the 'Repository URL' as indicated.
3. Give the project a directory name, like test, Jim, or whatever you want.
4. Next, browse for the place on your desktop where you want this project to live, its subdirectory.
5. Click 'Create Project', and let the magic be done!

What you just did was tell RStudio to associate a new project in R with specific repository on GitHub.

Step four: Alternative

If you still haven't built your first repository on GitHub, we can do something slightly different here. In RStudio, click *New project* and then *New Directory*. Call it what you want and change the directory as needed, make sure to tick *Create a git repository*, and then click *Create Project*. This creates an `.Rproj` file, which you can manage in the usual way through RStudio, including adding `README.md` and `LICENSE.md` files as discussing in Task 1.

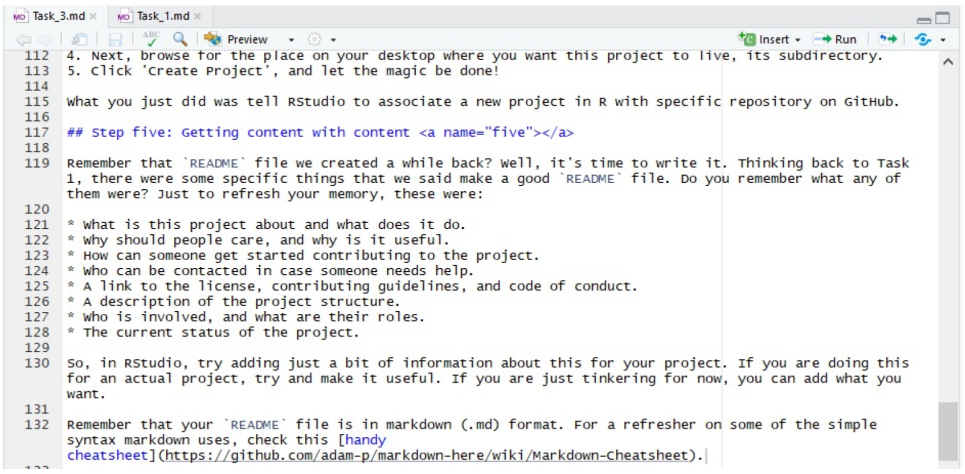
Step five: Getting content with content

Remember that `README` file we created a while back? Well, it's time to write it. Thinking back to Task 1, there were some specific things that we said make a good `README` file. Do you remember what any of them were? Just to refresh your memory, these were:

- What is this project about and what does it do.
- Why should people care, and why is it useful.
- How can someone get started contributing to the project.
- Who can be contacted in case someone needs help.
- A link to the license, contributing guidelines, and code of conduct.
- A description of the project structure.
- Who is involved, and what are their roles.
- The current status of the project.

So, in RStudio, open that file try adding just a bit of information about this for your project. If you are doing this for an actual project, try and make it useful. If you are just tinkering for now, you can add what you want.

Remember that your `README` file is in markdow n (.md) format. For a refresher on some of the simple syntax markdow n uses, check this handy cheatsheet.



```
4. Next, browse for the place on your desktop where you want this project to live, its subdirectory.
5. Click 'Create Project', and let the magic be done!

What you just did was tell RStudio to associate a new project in R with specific repository on GitHub.

## Step five: Getting content with content <a name="five"></a>

Remember that `README` file we created a while back? Well, it's time to write it. Thinking back to Task
1, there were some specific things that we said make a good `README` file. Do you remember what any of
them were? Just to refresh your memory, these were:

* What is this project about and what does it do.
* Why should people care, and why is it useful.
* How can someone get started contributing to the project.
* Who can be contacted in case someone needs help.
* A link to the license, contributing guidelines, and code of conduct.
* A description of the project structure.
* Who is involved, and what are their roles.
* The current status of the project.

So, in RStudio, try adding just a bit of information about this for your project. If you are doing this
for an actual project, try and make it useful. If you are just tinkering for now, you can add what you
want.

Remember that your `README` file is in markdown (.md) format. For a refresher on some of the simple
syntax markdown uses, check this [handy
cheatsheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet).
```

Screenshot of what this module looks in markdown, during development. Meta.

Step six: A brave commitment

OK, so now you should have a nicely edited `README` file. Now w e are going to 'commit' this to the project using Git. This is basically the equivalent of saving this version of your project, w ith a record of w hat changes w ere made. Successive commits produce a history that can be examined at a later time, allow ing you to w ork w ith confidence.

There are a few w ays of doing this.

1. Go to **Tools > Version Control > Commit**
2. In the environment pane in RStudio, there should be a new 'Git' tab. Handy.
3. In your console pane, there should now be a new 'Terminal', w hich you can run Git command lines through.

Let's just stick w ith the second option for now. This Git pane show s you w hich files have been changed and includes buttons for the most important Git commands w e saw earlier.

Select the `README` file in the Git w indow, w hich should show up automatically if you have made any edits to it. This adds that file to the 'staging' area, w hich is sort of like the pre-saving space for your w ork. Click 'Commit' and a new w indow should pop up.

Here, you have a chance to review your changes, and w rite a nice commit message. Type in something brief, but informative about the changes that you have made in this version or snapshot of your w ork. You w ant this to be enough information so that if you or someone else looks back on it, you'll know w hy you made this commit and the changes associated w ith it. These are like safety nets for your project in case you need to fall back for some reason.

> **Pro-tip**: Here, you w ill see a list of all the changes you have made since your last commit. Older removed lines are in red, and new ly added lines are in green. Double check these to make sure that the edits you have made are the ones you intended to make. This is really helpful for spotting typos, stray edits, and any other little mistakes you might have accidentially introduced. Safety first.

Note If you are colour-blind and can't see w hich lines have been added or removed, you can use the line numbers in the tw o columns on the left of the w indow as a guide. Here, the number in the first column identifies the older version, and the number in the second column identifies the new version.

Now w hen you click 'Commit', another w indow w ill pop up, telling you how many files you have changed and the number of lines w ithin that file you have changed. Close that little w indow dow n.

Step seven: PUSH!

Click the *Push* button in the top right of the new w indow. A new w indow w ill pop up now. What this is doing is synchronising the files changed on your local repository w ith the `README` file to the online version of the project on GitHub.

To do this from the Shell, use the follow ing command:

```
git push -u origin master
```

Some times here you w ill be prompted to add your username and passw ord from GitHub, w hich you should do if asked.

Close that w indow dow n, and the next one. Go to your project on GitHub, refresh, and check that the `README` file is still there in all its new ly edited glory. You should see the commit message you made next to the file too.

OPTIONAL ADVANCED/AWESOME STEP

Alright, so you just pushed some content to your first repo, awesome! Now let's put it into practice for a real project. Like, the one you are participating in right now. Let's try this out:

1. Go to the repositors for this project on GitHub

2. Fork the repository to your own GitHub account. The URL for this should be: `https://github.com/OpenScienceMOOC/Module-5-Open-Research-Software-and-Open-Source.git`

3. Head into RStudio, go to **File > New Project**, choose *Version Control*, select *Git*, and then paste the forkerd repository URL found in your copy of the repository. You now have your own versioned copy of this whole module. Neat. Save this somewhere on your local machine.

4. Now, you need to tell Git that a different version of this project exists. Open up the *Shell*, and enter the command: `git remote add upstream https://github.com/OpenScienceMOOC/Module-5-Open-Research-Software-and-Open-Source`

5. What you just did was name the original branch here `upstream`, just to keep things simple for now. Now, create a new **branch** to document your changes to this independent of the main branch. Enter the command: `git checkout -b proposed-changes master`

6. You just created a new branch called `proposed-changes` where you can now edit all of the content and files to your heart's delight. Hopefully, the structure of this project is simple enough for you to navigate around. All of the raw files for the MOOC can be found in the `content_development` folder, and this is `Task_3.md`.

7. If you scroll to the bottom of `Task_3.md`, you should see a place where you can edit in your name and affiliation. Add these in, and then go through the commit procedure detailed above. If you see anything else that needs editing too, feel free to add them in too!

8. Now, you want to push thee changes back to the original branch. Use the following command in your *Shell*: `git push origin proposed-changes`

9. Go back to GitHub and find your fork here. Click the little green button, and create a pull request. This is essentially a review to integrate the changes made into the original branch for this MOOC project.

10. The owners in charge of the MOOC project will now get a notification of this, review it, and confirm it if everything went to plan! We will review it, and if it all went okay, your name will now appear for all eternity as someone who completed this advanced task.

11. Have a cup of tea, coffee, or wine to celebrate!

CONGRATULATIONS

You just integrated Git with R Studio, and made your first change to a version controlled project. Your life will now never be the same, and your research workflow will probably be more rapid, agile, and collaborative than ever. Good luck going back to Word.

The great thing is that this doesn't have to just be used for code. You can use it for plain text, markdown, html, and, well, R code. The possibilities are limitless - what you have just learned is a new form of openly collaborative project management that works for an enormous range of tasks.

From now on, it is all up to you! Some advice is to:

- Make frequent commits. Treat Git like your puppy, in that it requires constant and special attention. Just a pat on the head every now and then is enough to keep it satisfied, but it'll be happiest with sustained servicing.

- The best way to do this is to make a commit each time you work on a specific problem. For example, writing a paragraph, running an analysis, or fixing a bug.

- Push often. Don't let those commits build up, otherwise you run more risk of getting into merge conflicts. Seeing as these can be the stuff of nightmares, just make sure to push often.

- Pull often. If others are working remotely on the same project, you will want to stay up to date with their changes. Make sure to frequently pull in their changes from GitHub to make sure you are all in sync.

- Experiment and explore! This task really only scratches the surface, and there are many different functions, tools, and ways this can be used. Really, it is up to you to find out how to use this information to improve your research workflow, and ultimately collaborate on better, more open and reliable research!

- To learn more about issues, branches, merge conflicts, pull requests, and other advanced aspects of using Git and RStudio, check out this awesome guide by Hadley Wickham.

Know a way this content can be improved?

Time to take your new GitHub skills for a test-run! All content development primarily happens here. If you have a suggested improvement to the content, layout, or anything else, you can make it and then it w ill automatically become part of the MOOC content after verification from a moderator!

List of participants who completed the ADVANCED version of this task

- YOUR NAME AND AFFILIATION HERE
- YOUR NAME AND AFFILIATION HERE