Manuscript

1. Introduction

2. Setting Up a Collaborative Workspace

- How to create a repository in Git Hub
- How to link it to Rstudio
- How to create a project in your local computer and export it to GitHub
- Basic commands for Git and GitHub fromt he RStudio terminal

A collaborative workflow requires a collaborative work space that enables everyone participating to share and contribute to a project.
There are multiple options for such a workspace, like Teams, Discord, or even over e-mail. When working with code however, online repositories on GitHub can be a good alternative.

In this tutorial we assume that you will be working with RStudio and have already downloaded R and Rstudio. In addition you are going to need to have the version control software Git installed and have an account in GitHub. If you need guidance for this you can find a helpful tutorial here.

What is a repository?

A repository is basically like a project box where you collect all the files, data, graphs and code scripts from your project.
Online repositories can be accessed from the internet and from any computer, while a local repository is only stored in a specific computer and cannot be accessed elsewhere. When setting up a collaborative work space its advantageous to have an online repository so that multiple people can contribute from their own computer to a shared repository, without having to send files by mail etc. In addition we can connect the online repository what a local repository which allows us to work and make changes using our own computer and then we can upload it to the online repository.

What is Git?

Git is a version control software that allows you to track the different version of your files. It basically allows you to keep a detailed history of changes you have done in your document and also what other people have added or removed in your collaborative documents. Having a version controls software set up for your workflow is very handy as it prevents major losses of documents and changes, and if any error is introduced in a document or code, you can track it back to see what and who submitted it. This fosters reproducibility, transparency, collaboration and robustness for your project.

What is GitHub?

GitHub is a collaborative online platform that allows you to host and join online repositories. its kinda like facebook for coding. gitHub allows us to share and collaborate with the people on the same code at the same time. It can also be used to host webpages and other stuff.

In this toturial we will only work with the RStudio interface and the online GitHub interface. however, if you want an expanded commandline and interface for GitHub you can use GitHUb CLI and/or GitHub Desktop. See toturials here: GitHub CLI and GitHub Desktop.

2.1 How to create a project that is connected between RStudio and GitHub?

When creating a new project and you want to link your local project with an online repository, you can go about it to ways basicly.
a) You can create the online repository and then clone it down to you computer
b) You can create a local repository and then push it online to GitHub

We´ll go through both options here, starting with the online repository.

Image file path:

resources/images/

2.1.1 Starting with an online repository

Step 1. Create a new online repository in GitHub (Video tutorial)

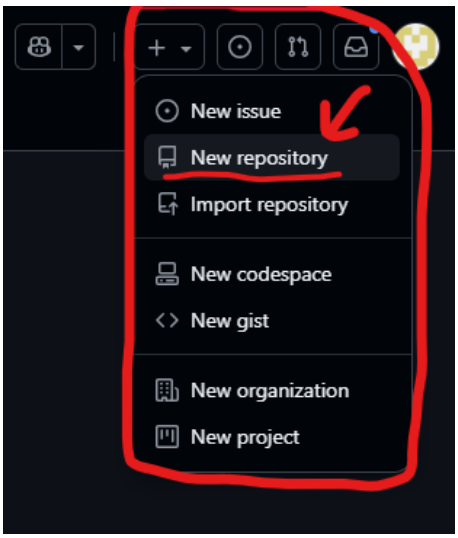

Figure 1: Creating a new online repository.

Once you´ve logged into GitHub, navigate to the top right corner of your page and find the + tab. Drop it down to reveal the "New repository" option. Click on it.

This will take you ta the repository creation page.
Here you give your repository a name, a description of what it will entail and wherever it is public or not.

You also have the options of adding a README file and a .gitignore file upon creation, but it is possible to create these after the repository is made as well.

README

A README file is a descriptive file that should explain what the project/repository is about, how it is organized and what the data in it means etc. Any additional information you want people to know when using your repository should go into the README.

.gitignore

The .gitignore file is an information file that tells Git what types of files it should track, or specifically not track. This is useful when you for example don't want to track the generated images or graphs from your code, but just your code.

So, now that you have created your first online repository you want to connect it to your local computer. You can do this multiple ways, but in this tutorial we´ll use commands in the terminal to initialize

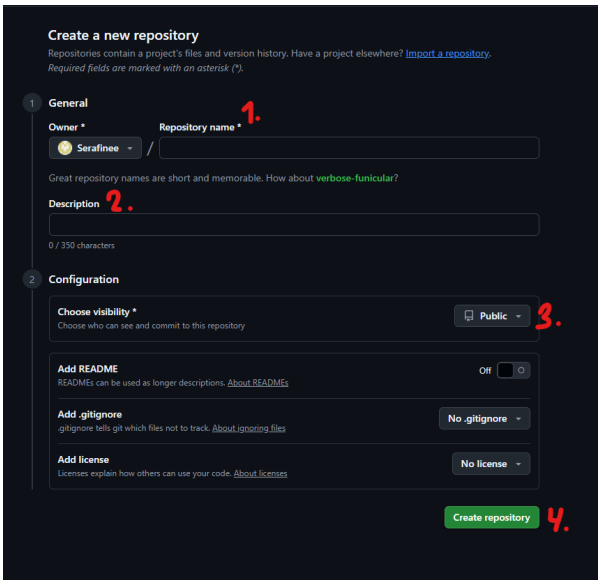

Figure 2: Repository setup-page.

Step 2: Copy the Repository URL

1. Go to your GitHub repository page.
2. Click the green **Code** button.
3. Copy the repository URL:
    - For example: `https://github.com/yourusername/repositoryname.git`

Step 3: Open RStudio and Clone the Repository

1. Open RStudio.
2. Go to **File** → **New Project** → **Version Control** → **Git**.
3. Paste the GitHub repository URL into the "Repository URL" field.
4. Choose a folder on your local computer where you want to clone the repository.
5. Click **Create Project**.when doing a commit on a file that has been staged, that version of the file goes into the version history. It is also tracked.

And tada! You have now cloned the online repository to your computer! Great job!

Step 4 (Optional): Configure Git in RStudio

If this is your first time using Git with RStudio, you'll need to configure your Git credentials. Open the RStudio terminal (**Tools** → **Terminal**) or navigate to the **terminal tab** at the top right of the RStudio interface and run the following commands:

```
git config -global user.name "Your Name"

git config --global user.email "your_email@example.com"
```

Replace the placeholder names in "Your Name" and "your_email@example.com" with your own.

Also, if Git has not yet been initiated in your RStudio project you can use the command:

```
git init
```

To check whet ever Git is initialized in your project you can write:

```
git status
```

If git is not initialized an error message will show up. Then just run the `git init` command and follow the instructions.

2.1.2 Starting with a local project in RStudio

Now, what if you wanted to do it the other way around, like if you already have a local project on your computer and want to create an online repository for it?

Step 1. Create a New Local Project in RStudio

If you already have a local project, you can skip this step. If not:

1. Open **RStudio**.
2. Go to **File > New Project > New Directory > New Project**.
3. Choose a folder where you want the project to live and give it a name.
4. Make sure to check the box **Create a Git repository**.
5. Click **Create Project**.

This initializes a local Git repository in your project directory.

If you already have made a project but it is not connected to a Git repository you can do it like this:

1. Navigate to **Tools > Project Options > Git/SVN**.
2. Select **Git** and click **Yes** when prompted to initialize a Git repository for your project.

Step 2. Create a New Repository on GitHub

1. Create a new repository in Git Hub like previously in section 2.1.1 step 1
    - Do **not** initialize the repository with a README, `.gitignore`, or license (we'll connect the existing local repository later).

You now have a new, empty GitHub repository.

Step 3. Link the Local project to the GitHub Repository

1. Copy the URL in the same way as previously in section 2.1.1 step 2.
2. Open the **Terminal** in RStudio (or use any terminal on your computer).
3. Navigate to your project folder, if you're not already there, by clicking on the dropdown menu in the top right corner of the RStudio interface and choose your project.
4. Add the GitHub repository as the remote origin using this command in the terminal:

```
git remote add origin https://github.com/yourusername/your-repository.git
```

5. Verify the remote connection using this command in the terminal `git remote -v`

You should be able to see something like this:

```
origin  https://github.com/yourusername/your-repo.git (fetch)
origin  https://github.com/yourusername/your-repo.git (push)
```

That means you have now successfully established a connection between your local project and the online repository on GitHub. Congratulations!

2.2 Workflow between local and online repositories

Now that we have connected our local repository with the online one, we can start to pushing some code! But before we jump into the commands for transferring files between our local and online repository, we should better understand how these processes work.

Take a look at the figure below:

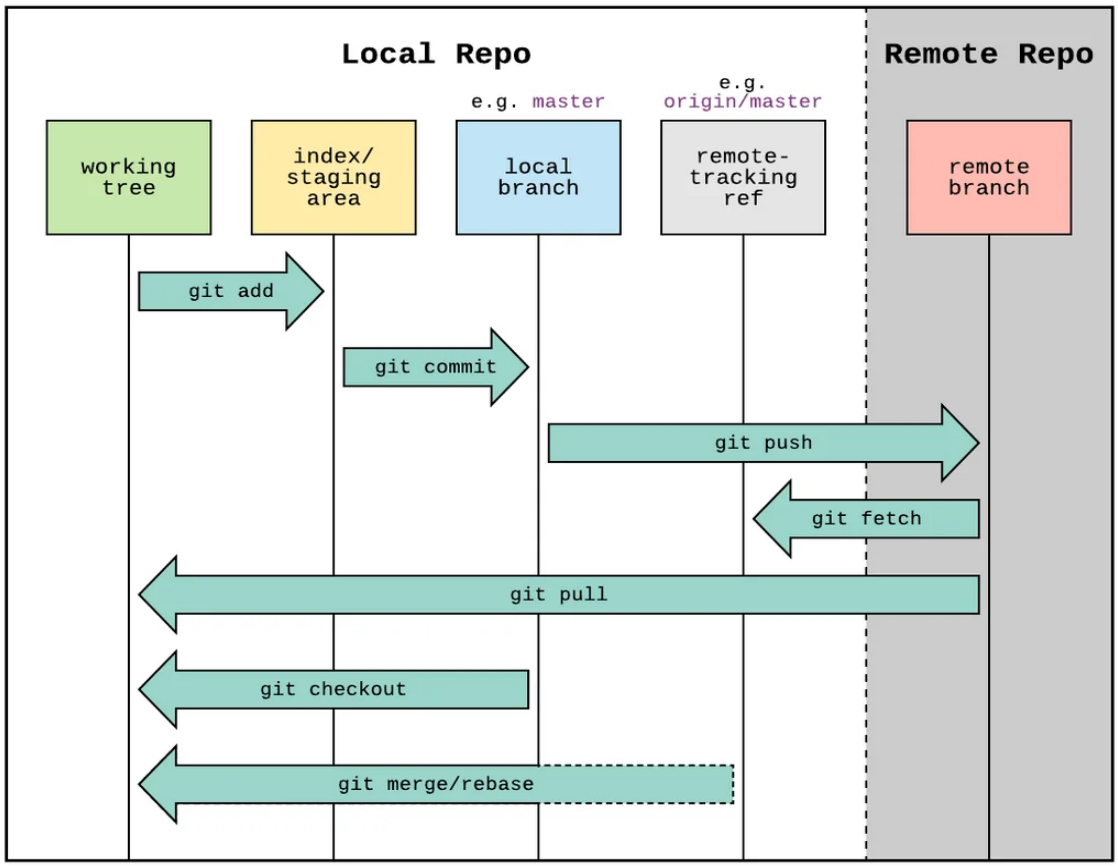

Figure 3: Visaluzation of local and online workflow in Rstudio.

Your **working tree** (also called **working directory**) is the project folder you are currently working in on your local computer. This is where you do all your edits on your files and code.

The **index/staging area** is a list is a list of files that Git is tracking for changes. When you "stage" files you tell Git to include these files in the next commit.

The **local branch** is a version of your project that exists entirely on your local computer. When you "commit" changes, you create a permanent snapshot of the files currently staged in the index and save them to your local repository.

The **remote tracking ref** is where you track the state of the online repository. When you "fetch" changes from the online repository, Git downloads the latest updates from the online repository but without merging it into your working directory. This allows you to see changes from collaborators or updates from the remote repository while keeping your working tree unaffected until you explicitly merge or rebase the changes.

2.2.1 Commands

Okay, now we can take a look at the commands we can use for this workflow.

Lets say you have edited some files in your working directory and want to add them to your **staging area** before committing them to your repository. If you only want to "stage" some files, but not all, you can use the command in the terminal:

```
git add file_name.txt
```

Just replace the file_name.txt with the name of the file that you want to stage. Remember to also include the file extentions (like .txt or .pdf etc.)

If you want to stage **all files in your repository** that have had changes to them you can write:

```
git add -A
```

This command stages **all changes** in the repository, including modified files, newly created files and deleted files.

Great! Now your edited files are ready to be committed! You can commit your files using this command:

```
git commit -m "Commit message"
```

When you run this commit command, all the files that you have staged are committed to your **local branch!** Its good to include a **descriptive commit message** that explains what your commit entails. That way its easier to track where changes or errors are introduced in your repository. Messages could be "Changed font headings" or "Added statistical ANOVA analysis to data processing".

Now that you have committed the files you want to push them to your remote directory using:

```
git push
```

This command uploads everything you have committed so far this session to your online repository and updates it based on your commits. Using this command you can push multiple commits at the same time.

Now lets say your colleague, who you are collaborate with, has added a new file to the online repository on GitHub. You want to pull that document from the online directory down to your local repositopry and start editing it. The most straightforward way is to use:

```
git pull
```

This command pulls the latest changes from your remote repository and merges them into your **current branch** in your local repository, so that they are exactly the same. Then you can just start editing and making changes.

If you don't want to fully copy the online repository yet, but take a look at it first before merging it ito your working directory you can use the command

```
git fetch
```

This command downloads the changes from the remote repository (e.g., **`origin`**) and updates your remote-tracking branches (e.g., **`origin/main`**) without modifying your working directory or local branch.

To look at the branch you have fetched you can use one of these two commands:

```
git log HEAD..origin/main

git diff HEAD..origin/main
```

git log will give you a list of the commits that are not in your local branch (**HEAD**) together with their metadata (authors, date etc.). This is helpful if you want to quickly answer "What was changed?" and "Who changed it?".

git diff will show you the actual changes that are between two points, ex. your local branch (**HEAD**) and your fetched branch (**origin/main**). git diff will show you the specific lines of code that were added modified or deleted in the fetched branch. This is helpful if you want to know exactly what was changed.

If you then want to integrate the fetched branch with your current branch, you can use the command:

```
git merge
```

Using this command, Git creates a **merge commit**, which combines the changes from both branches while preserving the complete commit history of both branches. The merge commit explicitly shows the point where the branches diverged and were brought together.

An alternative way to integrate the fetched branch is to use:

```
git rebase
```

This command does **not create a merge commit**. Instead, it **rewrites the commit history** of your current branch by replaying its commits on top of the fetched branch, creating a **linear history**. Essentially, it re-aligns your current branch so that it starts from the latest commit of the fetched branch, as if your changes were made after the fetched branch's changes. This results in a cleaner, more linear commit history.

Questions:

- add sections about README files and .gitignore
- should I add a section about the concole and the terminal andhow they are used? Or could this perhaps go into the Introduction?
- Should we include stuff about creating branches?

Disclaimer: This section was written with the help of SIKT KI chat using the gpt-4o model. The AI was used to verify code chunks, summarize steps in an organized format and rewrite original text for better grammar and flow. The author takes full responsibility for the resulting output. Link to AI model.

3. **Conducting Simulations Before Data Acquisition**

4. **Including Data Packages for Distribution**

5. **Creating Visualizations from Data Packages**