

gesis

Leibniz Institute  
for the Social Sciences



# Workflows for Reproducible Research with R & Git

## Git & GitHub - Part 2

*Johannes Breuer, Bernd Weiss, & Arnim Bleier*

*2023-11-16*

# Interacting with Git

There are various tools that we can use for interacting with Git. There are command line interfaces (CLI), such as *git bash* for *Windows* or the Terminal on *MacOS*.

There also are Graphical User Interfaces (GUI), such as *GitHub Desktop* or *GitKraken* (for an overview of Git clients with a GUI, see <https://git-scm.com/downloads/guis>).

# A note on tool stacks

While we introduce you to different tools for reproducible research in this workshop and it is always possible to "mix and match", it is usually advisable to try to minimize the number of different tools in your workflow.

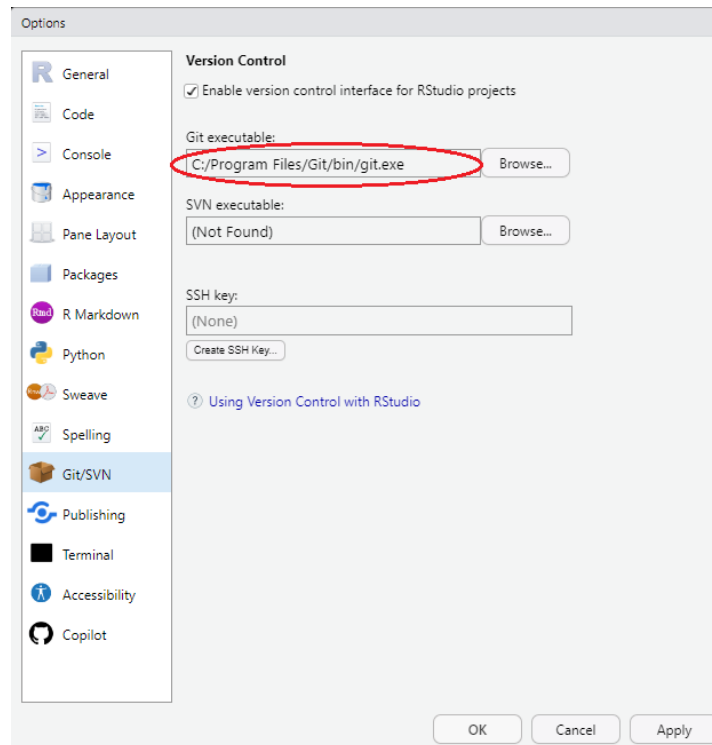
# VC in your favorite IDE

Lucky for us, the most popular Integrated Development Interfaces (IDE) for  $\mathbb{R}$ , *RStudio*, also offers functionalities for using `Git`.<sup>1</sup>

[1] Another popular IDE that works nicely with  $\mathbb{R}$  and `Git` is *Visual Studio Code* - or *VSCo* for short - by *Microsoft*. *VSCo* offers an extension for  $\mathbb{R}$  and also works nicely with *GitHub* (which is not surprising, given that both are *Microsoft* products).

# All set?

If you have correctly set up `Git`, *RStudio* should be able to detect it.  
The easiest way to check this is via the *RStudio* options: *Tools -> Global Options -> Git/SVN*



# Finding Git

If you have properly installed Git and *RStudio* cannot detect your local Git executable, you need to tell it where to find it via the *Browse* button in the menu shown on the previous slide.

On macOS and Linux, a common path for the Git installation is (something like) `/usr/bin/git`, while on Windows it is (something like) `C:/Program Files/Git/bin/git.exe`.

*Note:* You should restart *RStudio* after making changes in this menu.

# How to use `Git` in *RStudio*

There are essentially two options for using `Git` via *RStudio*

1. Through the GUI
2. Via the Terminal

# Other options for using `Git` with `R`

While we won't cover those in any detail in this session, there are also `R` packages for `Git` operations:

- `usethis` can, e.g., be used to initialize a `Git` repository or for managing credentials
- `gert` is a simple `Git` client for `R` that can be used to perform basic `Git` commands, such as staging, adding, and committing files, or creating, merging, and deleting branches
- `ghstudio` provides some "experimental tools to use git/github with RStudio, e.g see issues and diffs in the viewer"<sup>1</sup>

[1] As you can see in the commit history of the *GitHub* repo for this package, there has been no active development since April 2022, so the current functionality and future of this package may be unclear.



# Are you legit to `Git`?

As a reminder, there are two protocols for securely communicating with remote `Git` servers, such as *GitHub*: `HTTPS` and `SSH`.

We will use `HTTPS` with a Personal Access Token (PAT).

# Creating a PAT

You should have created a PAT in preparation for this course (via the *GitHub* web interface). If you have not yet done so, you can also use a function from the `usethis` package.

```
library(usethis)  
create_github_token()
```

**NB:** Do not close the browser window/tab with the PAT until you have stored it somewhere. You should treat the PAT like a password.

# Storing a PAT

Once you have created a PAT, the simplest way to store it for use with R and *RStudio* is the `gitcreds_set()` function from the `gitcreds` package.

```
library(gitcreds)
gitcreds_set()
```

*Note:* In the background, this function uses the `Git credentials helper`. Of course, you can also (or additionally) store your PAT in another (safe) place, such as your password manager.

To check if setting the PAT worked, you can run the following command:

```
gitcreds_get(use_cache = FALSE)$password
```

## Storing a PAT

If, for some reason, `gitcreds_set()` does not work for you, you can also store your PAT in a **.Renviron file**. You can use a function from the `usethis` package for this:

```
usethis::edit_r_environ()
```

*Note:* There can be user- R-version- or project-specific `.Renviron` files. The function above opens the user-specific one which should be stored in your `HOME` directory. **NB:** Storing the PAT in an `.Renviron` file is less safe as this is a plain text file.

$$\text{Git} + RStudio = \img alt="red heart icon" data-bbox="641 141 698 210"/>$$

Now you should hopefully be all set to use `Git` as well *GitHub* via *RStudio*.

In order to get the best out of the combination of  $\mathbb{R}$ , *RStudio*, and `Git`, it is recommendable to adopt a "project-oriented workflow".

## Excursus: *RStudio* projects

*RStudio* projects are associated with `.Rproj` files that contain some specific settings for the project. If you double-click on a `.Rproj` file, this opens a new instance of *RStudio* with the working directory and file browser set to the location of that file.

*Note:* The repository/folder for this workshop contains an `.Rproj` file, if you want to try this out.

Explaining *RStudio* projects in detail would be too much of a detour at this point, but if your interested in that, you can check out the [RStudio support site](#) or the [respective chapter in \*What They Forgot to Teach You About R\*](#).

# What comes first?

In your everyday work, you quite likely need different workflows depending on the temporal order in which things are created or set up: your local project/files, version control with `Git`, and the remote *GitHub* repository.

- New project, GitHub first
- Existing project, GitHub first
- Existing project, GitHub last

The *New project, GitHub first* approach is the simplest. Hence, if possible, you should aim for this route. For the exercises in this session, however, we will build on what you did in the previous session and follow the *Existing project, GitHub last* approach.

# Git through the GUI

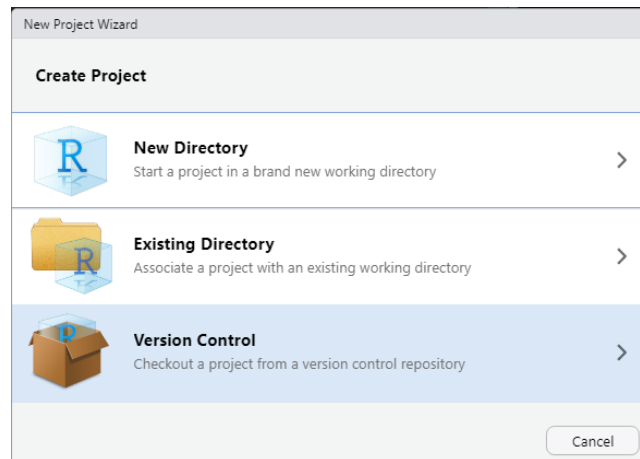
You can perform quite a few `Git` operations via the *RStudio* GUI: You can, e.g., create a new `Git` repository, clone an existing repository, stage and commit changes and push them to a remote repository, or pull changes from there, and merge those with your local changes.

We'll go through a few of these common steps in the following.



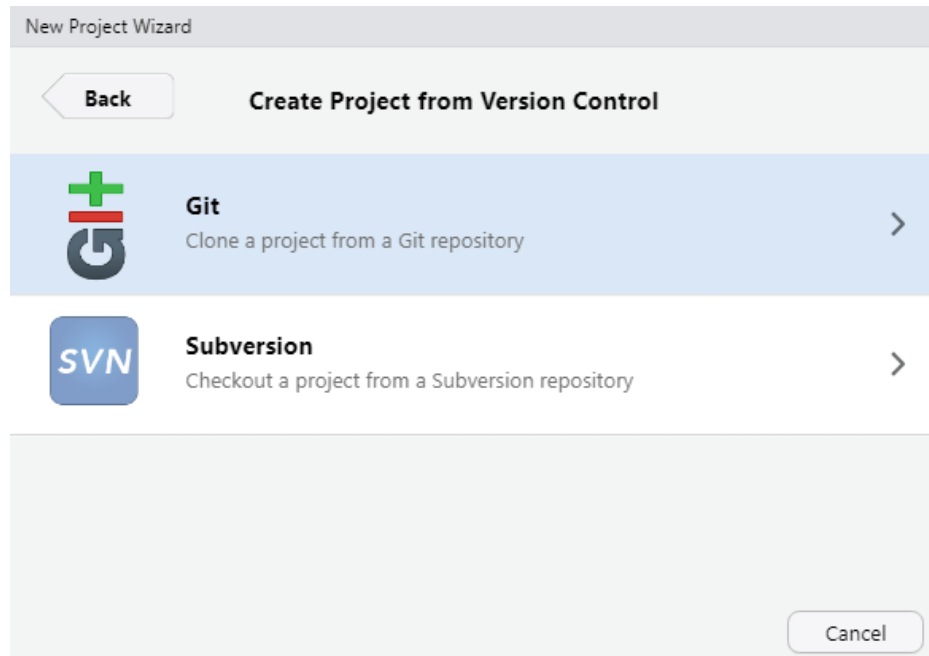
# New Git project connected to existing remote repo

You can create a new version-controlled project that is connected to a remote repository that already exists on *GitHub* via *File -> New Project -> Version Control* in the *RStudio* menu.



# New Git project connected to existing remote repo

Next, choose Git...



# Associate remote repo: HTTPS


In the menu that opens after that, enter the URL of the remote repository, give the local repository a name, and tell *RStudio* where it should be stored. It usually makes sense to check "Open in new session".

**NB:** Which URL you should enter depends on the authentication method you use. If you use `HTTPS`, you can simply copy the URL from the address bar of your browser.

# Associate remote repo: HTTPS

New Project Wizard

Back **Clone Git Repository**



Repository URL:

Project directory name:

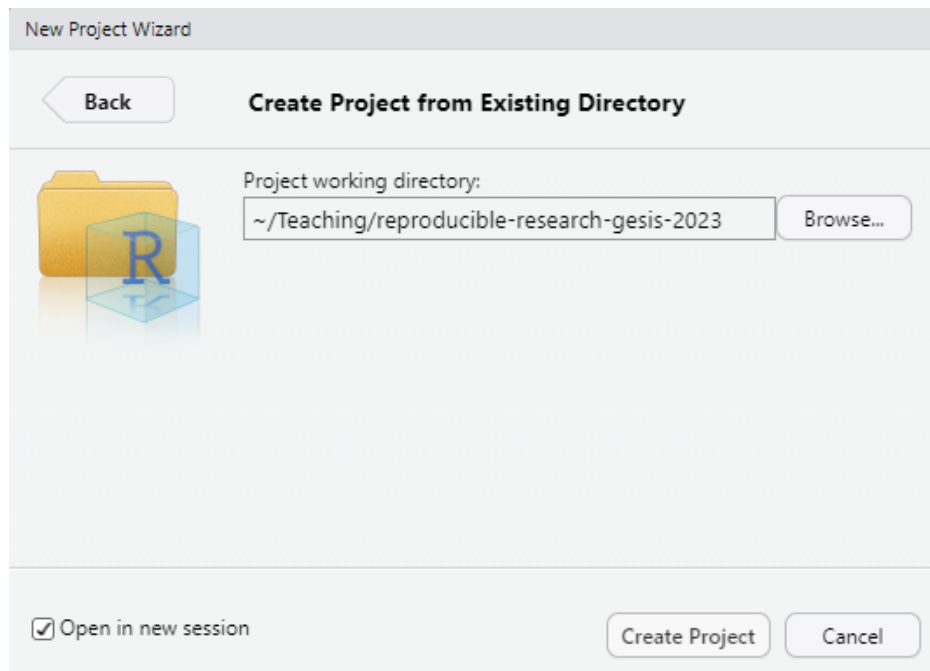
Create project as subdirectory of:

☒ Open in new session

Create Project Cancel

## Existing directory

If you want to create a new *RStudio* project ( `.Rproj` file) in an existing directory, you can also do that via the *RStudio* GUI: *File -> New Project -> Existing Directory*



## Existing directory

If you have already initialized `Git` in that directory, the `Git` tab should be visible in the newly opened *RStudio* instance and you're good to go. If not, you have three options of doing so:

1. Run the R command `usethis::use_git()` (in the console)
2. Via the *RStudio* GUI (in the new project): *Tools* -> *Project Options* -> *Git/SVN* -> select `Git` as the version control system (and confirm)
3. Run `git init` in the Terminal (we'll get to using the Terminal in *RStudio* in just a bit)

**NB:** If you still need to initialize `Git` in the directory, you will also need to add and commit all of the existing files that are in that directory (at least the ones that you want to be tracked; remember that you can add files that you don't want to be tracked to the `.gitignore` file).

## Existing directory

If you want to associate your local version-controlled project with a remote *GitHub* repository, you need to create one and connect it with your local project/directory. The easiest way to do this is using a function from the `usethis` package.

```
usethis::use_github(private = TRUE)
```

You can also [go through this process manually](#) through the *GitHub* web interface and *RStudio*.

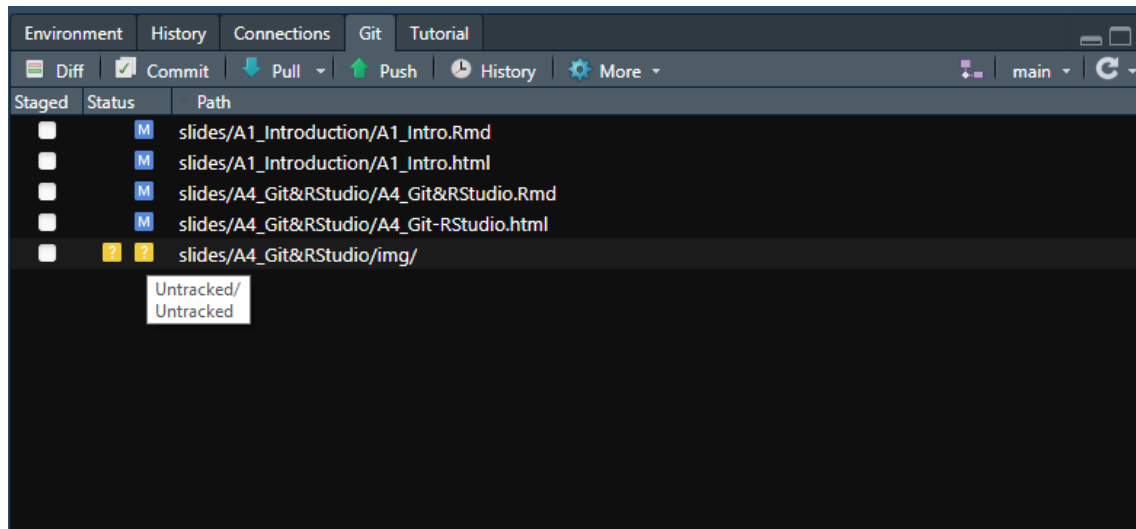
# Modifying & creating files within the project

Once you have successfully created a project (and connected it to a remote repository), you can start editing files or creating new ones. When you have modified existing files and/or created new ones and saved the changes, these will be displayed in the `Git` tab in *RStudio* and their status will be indicated as *modified* or *untracked*.

*Note:* The `Git` tab also displays in which branch you are currently working.

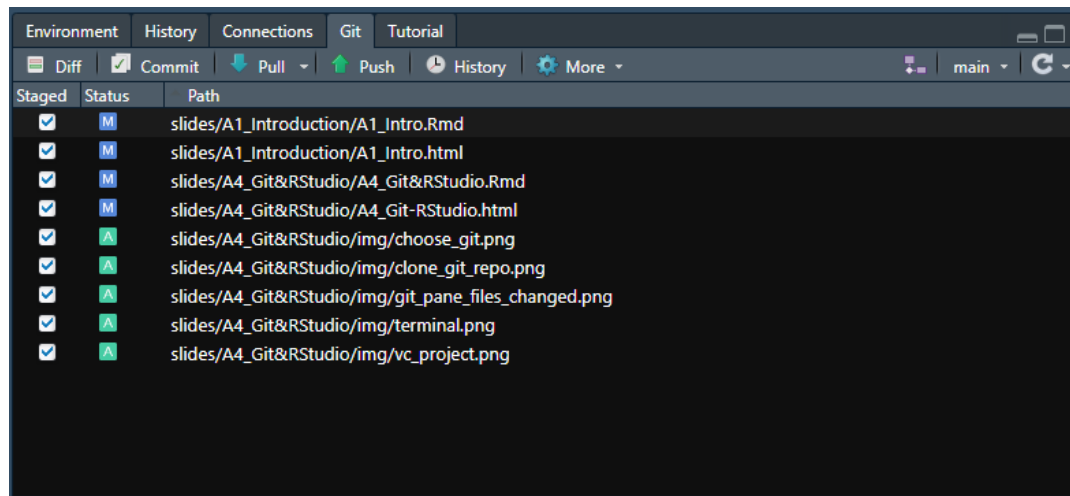


# Modifying & creating files within the project



# Staging changes

Once you have reached a point at which you want to commit (and possibly also push) your changes, you can stage them by checking the boxes in the *Staged* column in the `Git` tab. This the *RStudio* GUI equivalent of `git add`. The status of previously untracked files will then change to *added*.

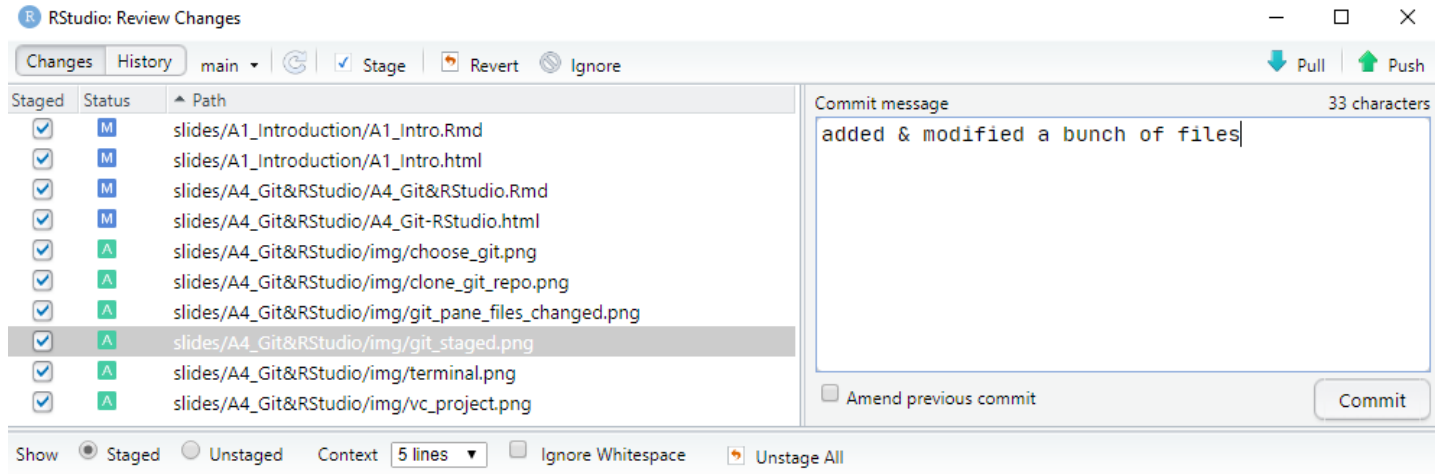


# Committing & pushing changes

After staging your changes, you can commit them via the *Commit* button in the `Git` tab. In the commit menu that opens you should enter a meaningful commit message. Once that is done you can click the *Commit* button.

If you want to, you can also directly push your changes to the remote repository on *GitHub* via the *Push* button. You can, of course, also do this at a later point (directly via the `Git` tab).

# Committing & pushing changes



# Brief notes about data protection

*IMPORTANT:* Make sure that you do not (accidentally) publish anything that should not be public via *GitHub*!

For our use cases, this will probably mostly concern...

- a) research data (esp. if it is on the individual level and contains personal information; licenses and usage agreements also have to be taken into account here)
- b) sensitive information like passwords, API keys, PAT/SSH keys, etc.

*Note:* If you are worried that a particular secret, such as an API or SSH key might have been compromised, you can use the [GitGuardian service](#) to check this. And if it is a *GitHub* PAT, you can check the [documentation](#) for how to revoke it.

# Brief notes about data protection

There are a few ways in which you can substantially lower the risk of accidentally sharing information or files that should not be public:

- Add files and folders you do not want to be tracked (and, thus, also not push to *GitHub*) to the `.gitignore` file for your project.
- Keep sensitive information/files in a separate place (i.e., not in the version-controlled project directory).
- When you create a new *GitHub* repository, set it to "private" and only set it to "public" once you are sure that it does not contain anything that should not be public.

*Note:* You can **remove data and (parts of) your commit history from *GitHub***, but it is best to employ the prevention measures listed above (also because there are bots that automatically clone new repos).

# Pulling changes from the remote repository

You can also pull changes from the remote repository via the *Pull* button in the `Git` tab.

As a general workflow recommendation (especially if you're just getting started with `Git` and *GitHub*) it is usually advisable to first pull from the remote repository before making (and then staging, committing, and pushing) any local changes. This is even more relevant when you collaborate with others on the same repository (more on that later).

# Pulling changes from the remote repository

**Important technical note:** If you click the *Pull* button in *RStudio* this will perform a **pull with rebase**. Put briefly, pulling with rebase means that local changes are reapplied on top of remote changes. This is **different from pull with merge**. In many scenarios, this is generally the preferable method and nothing you need to worry about. However, in some cases, this can cause issues, and it is good to be aware of this.



# Limitations of the GUI

While the *RStudio* GUI can be used for quite a few basic `Git` operations, it has a set of limitations. The first one is the use of specific defaults as is the case with pulling (with rebase). Another one is that it can become quite tedious to stage a large number of files through the GUI.

# Limitations of the GUI

Another downside of interacting with `Git` through the *RStudio* GUI is that there is a risk of *RStudio* becoming really slow or even crashing if you add/commit a lot of files at the same time and/or very large files. If the overall size of added or altered files is large, the Commit menu in *RStudio* usually also gives a warning about this.

*Note:* The *RStudio* GUI is also not the best tool for **handling merge conflicts**.

# Destination Terminal

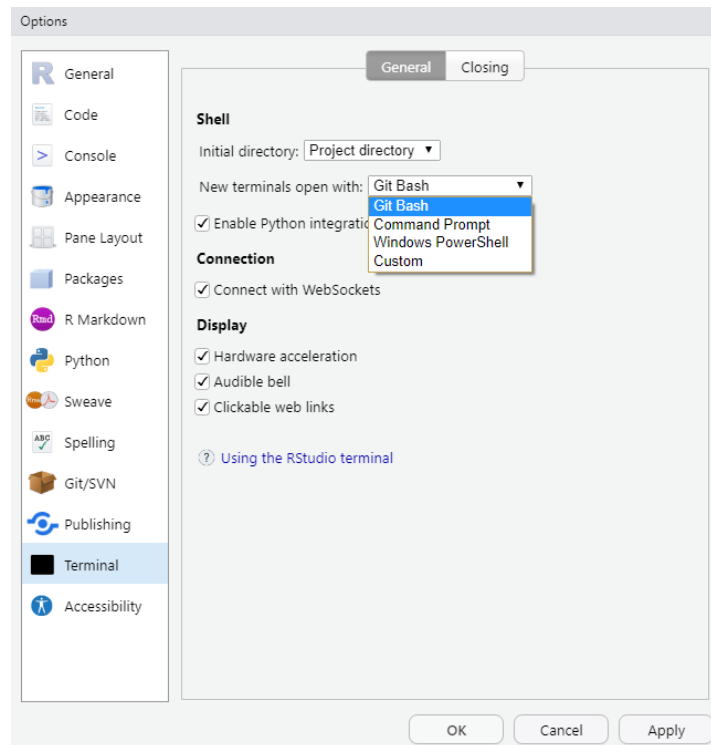
If you want to add/commit a lot of files or large files, want more control over the `Git` commands, or need to use more advanced `Git` operations, the *RStudio* GUI is not the right choice. Instead, you should use a command line interface (CLI).

# Destination Terminal

Lucky for us, if you need a CLI for using `Git`, you don't need to leave *RStudio*. As of version 1.3.1056-1, *RStudio* provides a `Terminal` tab in the console pane. Through this, *RStudio* provides access to the **system shell**. If you have properly installed `Git` you can use this to execute the full range of `Git` commands.

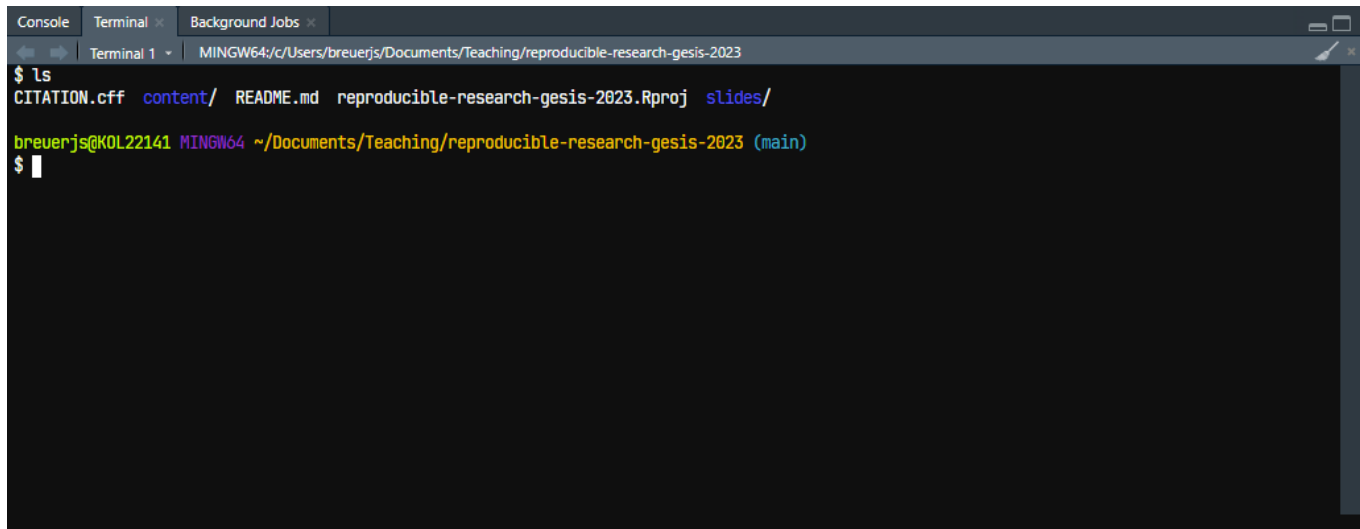
# Picking shells

Depending on your OS as well as your installation of `Git`, you can pick different shells to be run in the *RStudio* Terminal tab. You can choose those via the the *Terminal* menu in the *RStudio* Global Options.



# Picking shells

If you use *Windows* and have installed `Git` for *Windows*, you should use `Git Bash` as the shell that is run in the *RStudio* terminal (shown in the picture below). On *MacOS*, you should be able to simply use its own `Terminal` in *RStudio*.



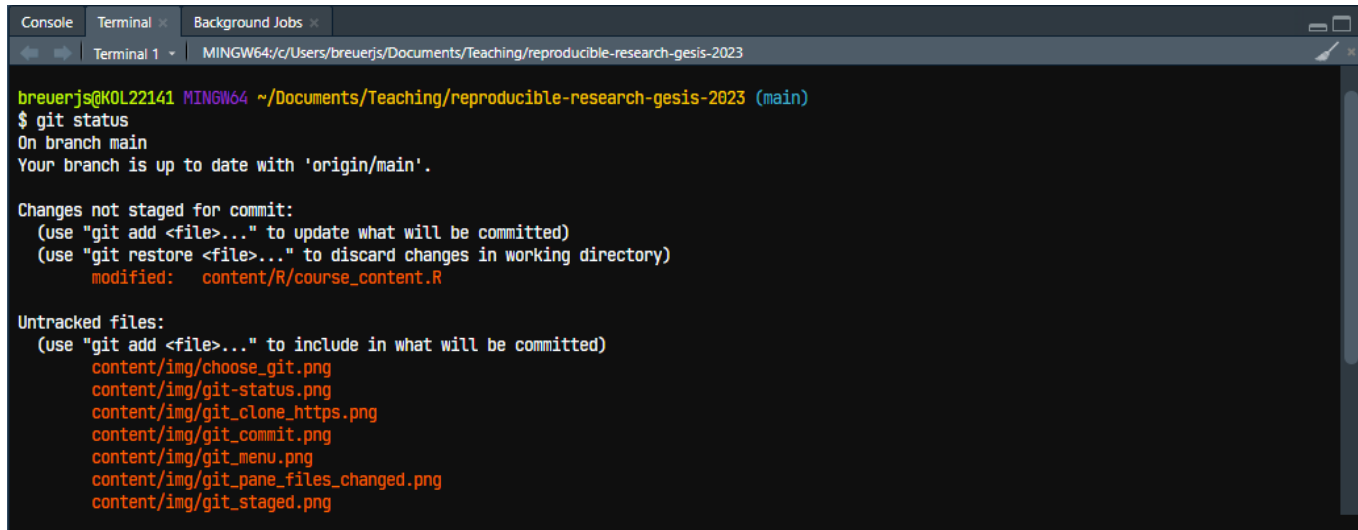
The screenshot shows an RStudio interface with three tabs: 'Console', 'Terminal', and 'Background Jobs'. The 'Terminal' tab is active, showing a terminal window titled 'Terminal 1' with the path 'MINGW64:/c:/Users/breuerjs/Documents/Teaching/reproducible-research-tesis-2023'. The terminal output shows the command '\$ ls' followed by the directory listing: 'CITATION.cff content/ README.md reproducible-research-tesis-2023.Rproj slides/'. Below this, the terminal shows the user 'breuerjs@KOL22141' on a 'MINGW64' system, in the directory '~/Documents/Teaching/reproducible-research-tesis-2023' on the 'main' branch. The prompt '\$ ' is visible at the bottom of the terminal window.

# Terminal and Shell in *RStudio*

For some more information on choosing and using the shell in *RStudio*, you can check out the [chapter on this in \*Happy Git and GitHub for the useR\*](#) or the *RStudio* How To Article on [Using the \*RStudio\* Terminal in the \*RStudio\* IDE](#).

# Using the Terminal in *RStudio*

You can use the Terminal in *RStudio* to run all available Git commands, such as `git status`.



```

Console | Terminal | Background Jobs
Terminal 1 | MINGW64/c/Users/breuerjs/Documents/Teaching/reproducible-research-gesis-2023

breuerjs@KOL22141 MINGW64 ~/Documents/Teaching/reproducible-research-gesis-2023 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   content/R/course_content.R

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        content/img/choose_git.png
        content/img/git-status.png
        content/img/git_clone_https.png
        content/img/git_commit.png
        content/img/git_menu.png
        content/img/git_pane_files_changed.png
        content/img/git_staged.png
  
```



# Using the `Terminal` in *RStudio*

You can also use the full range of arguments to customize your `Git` commands. For example, to stage and commit all changes, you could run the following command in the `Terminal`:

```
git add -A && git commit -m "Your Message"
```

# Notes on the exercises in this session

In the exercises for this session, you should populate your *GitHub* repository with some R content. You can, of course, use or write your own script(s) for this. However, we have also created a `demo_script.R` in the `exercises` folder within the workshop materials that you can use and build upon.

*Note:* The demo script makes use of the following packages: `dplyr`, `ggplot2`, `scales`, `correlation`, and `sjPlot`. In case you want to run the script, you need to install these packages first.

```
install.packages(c("dplyr", "ggplot2", "scales", "correlation", "sjPlot"))
```

# Notes on the exercises in this session

The `demo_script.R` makes use of some toy data that we included in the course materials (in the `data` folder). These are synthetic data based on data from waves "ja" and "jc" from the **GESIS Panel** created using the **synthpop package**.

The `data` folder also contains a **mini-codebook** (in the subfolder `doc`) as well as the files used for creating the synthetic data (in the spirit of reproducibility 😊).

**NB:** As mentioned in this session, you should NEVER publish real survey data on *GitHub*.

Exercise time 🏋️ 💪 🏃 🚴

Solutions

# Resources

A really great resource on using `Git` (and *GitHub*) in combination with `R` and *RStudio* is the website *Happy Git and GitHub for the useR* by **Jennifer Bryan**. Much of the content in this session is based on this resource and it offers a lot of additional helpful information and advice (including some help with troubleshooting commonly encountered issues).

Another good introductory resource is the *RStudio/posit* How-To Article on *Version Control with Git and SVN*.

# *Next up: Advanced operations with Git and GitHub*