

Reproducible research workflows for psychologists

Git & RStudio

Johannes Breuer & Frederik Aust

KU Leuven, 27.-28.04.2022

Interacting with `Git`

There are various tools that we can use for interacting with `Git`. Besides 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.

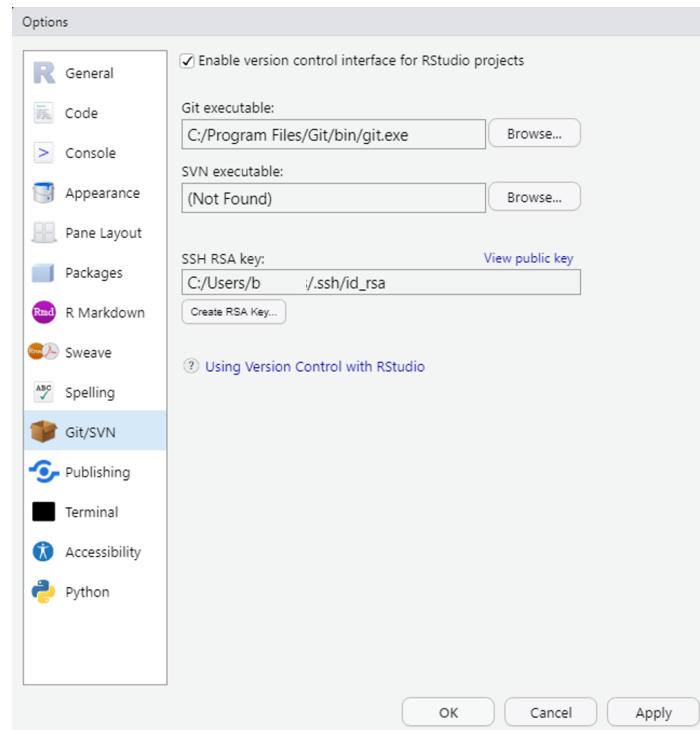
VC in your favorite IDE

Lucky for us, the most popular Integrated Development Interfaces (IDE) for `R`, *RStudio*, also offers functionalities for using `Git`.¹

[1] Another popular IDE that works nicely with `R` and `Git` is *Visual Studio Code* - or VSCode for short - by *Microsoft*.

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` is a novel in-development package that provides "experimental tools to use git/github with RStudio, e.g see issues and diffs in the viewer"

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`.

For our *GitHub* examples, we will use `HTTPS` with a Personal Access Token (PAT). For the *KU Leuven GitLab*, we will use `SSH`.

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.

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.

Note: Of course, you can also (or additionally) store your PAT in another (safe) place, such as your password manager.

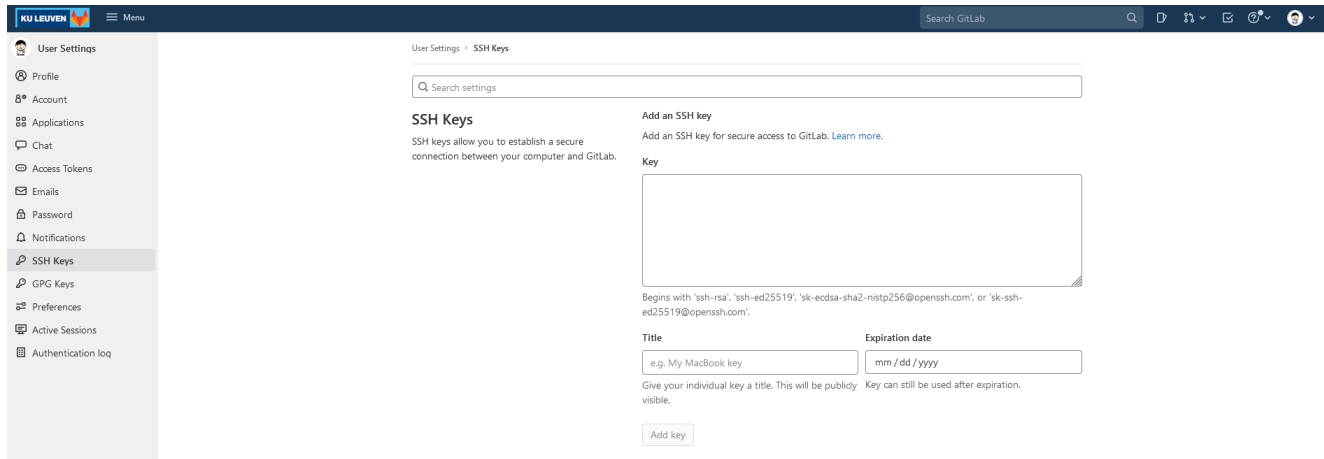
SSH

SSH stands for Secure Shell and is another option for secure interaction with remote **Git** servers, such as *GitHub* or *GitLab* instances. As for the PAT, you should have set up SSH in preparation for the workshop. If you have done so, the location of your RSA key should be displayed in the *RStudio* **Git** menu (*Tools* -> *Global Options* -> *Git/SVN*).

SSH

SSH

If you have not done so before, you first need to create an RSA key via the **Git** menu in *RStudio*. After that, you have to need to copy the public key into the key field in the SSH Keys page on *GitLab*.



Git + *RStudio* = 

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

In order to get the best out of the combination of **R**, **R Markdown**, *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.

Excursus: *RStudio* projects

Using *RStudio* projects can facilitate several things: the organization of files, the use of (relative) file paths, but also the integration of `R` and `R Markdown` with `Git`.

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

In this session, we will focus on the *New project, GitHub first* approach (which works the same for *GitLab*).

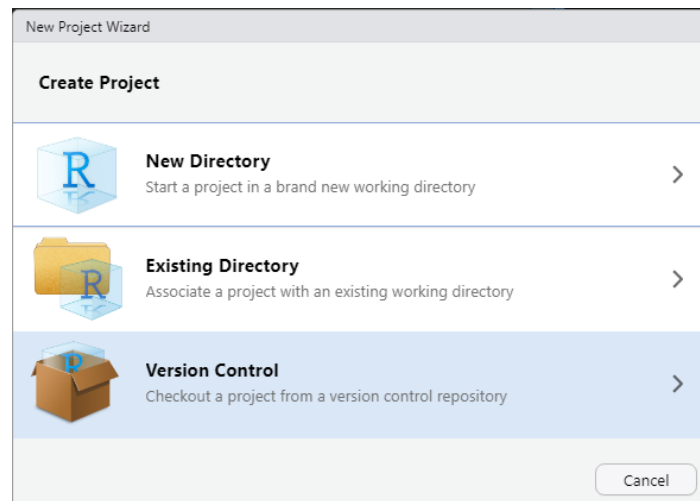
Git through the GUI

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.

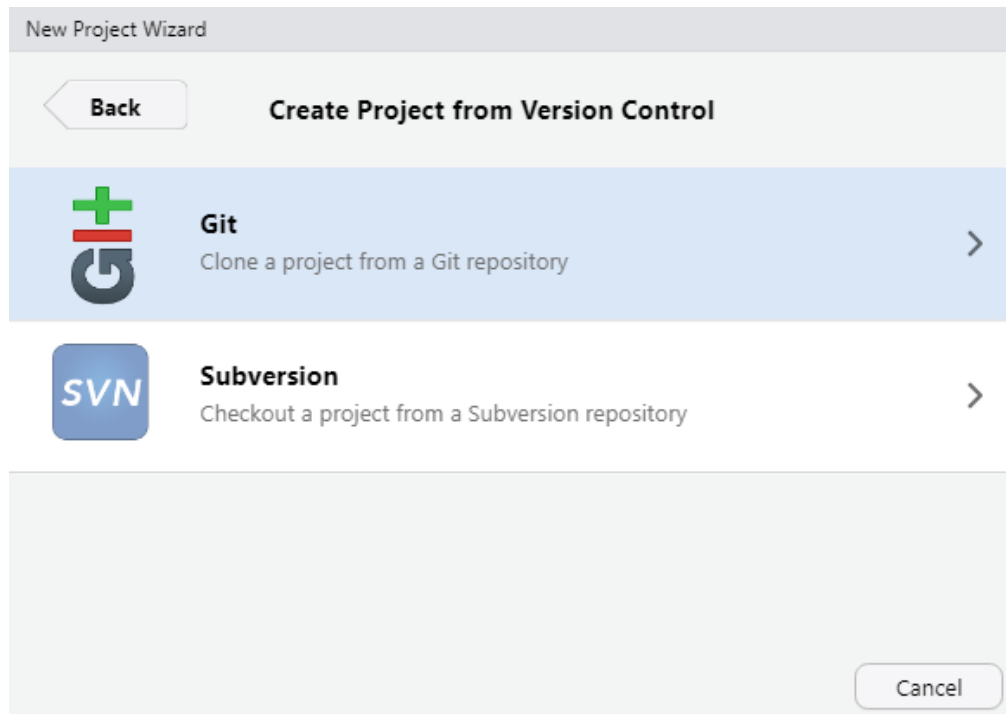
New **Git** project connected to existing remote repo

You can create a new version-controlled project that is connected to a remote repository that already exists on *GitHub* or *GitLab* 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:

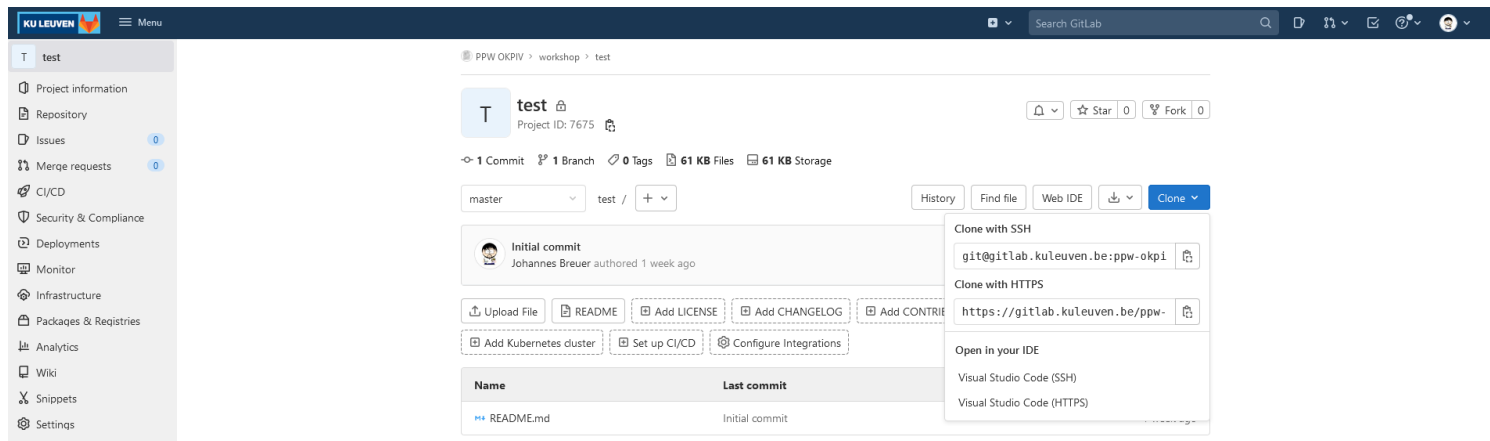
Create project as subdirectory of:
 [Browse...](#)

Open in new session

[Create Project](#) [Cancel](#)

URL for SSH

In case you use **SSH**, you can get the right URL from *GitLab* via the blue *Clone* button on the website of the repository.



The screenshot shows the GitLab interface for a repository named 'test'. The 'Clone' button is highlighted, and a dropdown menu is open, showing the following options:

- Clone with SSH: `git@gitlab.kuleuven.be:ppw-okpi`
- Clone with HTTPS: `https://gitlab.kuleuven.be/ppw-`
- Open in your IDE: Visual Studio Code (SSH), Visual Studio Code (HTTPS)

The repository page also displays the following information:


- Project ID: 7675
- 1 Commit, 1 Branch, 0 Tags, 61 KB Files, 61 KB Storage
- Initial commit by Johannes Breuer, authored 1 week ago
- Buttons for Upload File, README, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Add Kubernetes cluster, Set up CI/CD, and Configure Integrations.
- Table of commits:

Name	Last commit
README.md	Initial commit

Associate remote repo: SSH

New Project Wizard

[Back](#) **Clone Git Repository**



Repository URL:

Project directory name:

Create project as subdirectory of:
 [Browse...](#)

Open in new session

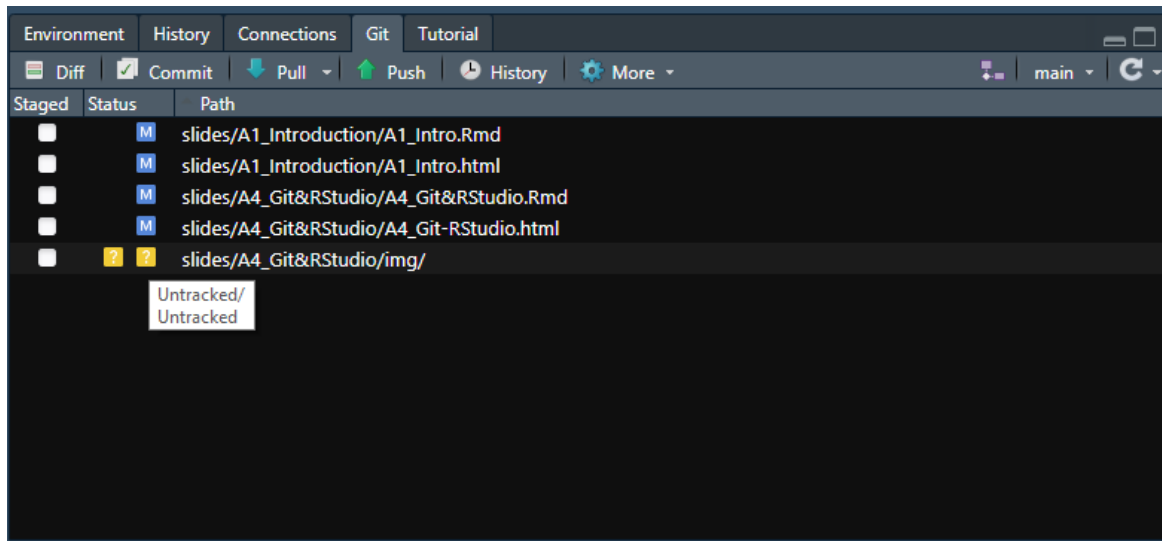
[Create Project](#) [Cancel](#)

Modifying & creating files within the project

Once you have successfully created the project, 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*.

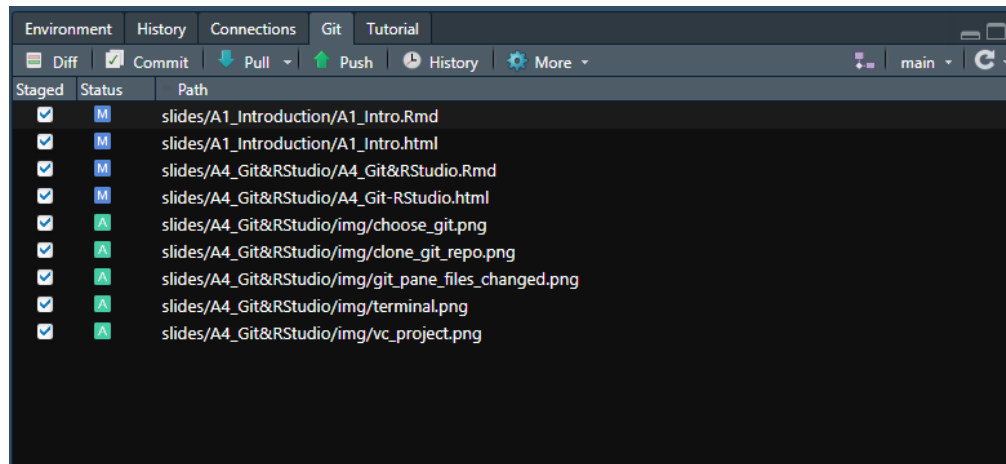
Note: The `Git` tab also displays in which branch you are currently working.

Modifying & creating files within the project



Staging changes

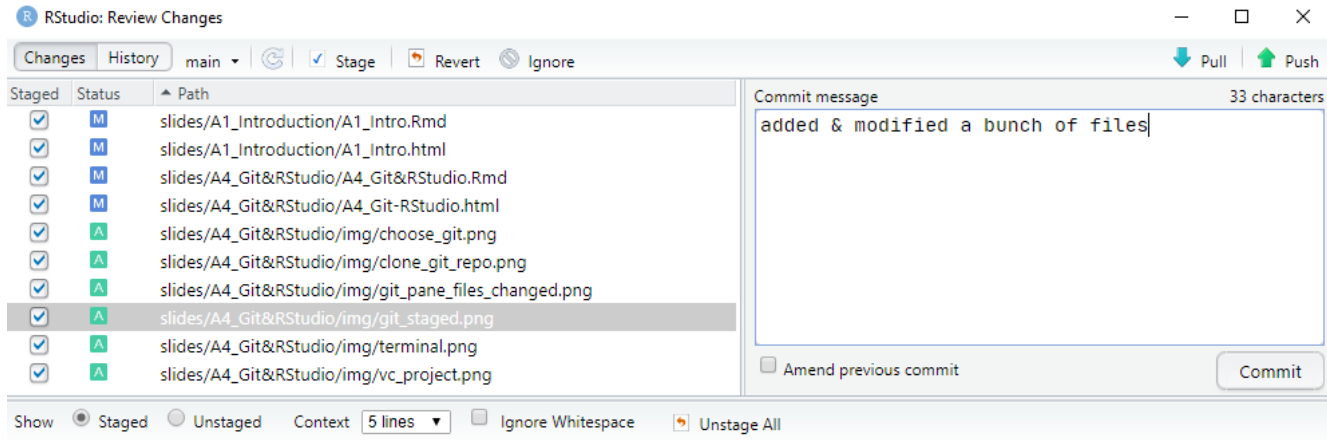
Once you have reached a point at which you want to commit (and possibly also push) your changes, you can stage them by checking the boxes in the *Staged* column in the `Git` tab. This is the *RStudio* GUI equivalent of `git add`. The status of previously untracked files will then change to *added*.



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



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/GitLab*) 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.

The *RStudio* GUI is also not the best tool for **handling merge conflicts** (we will discuss those in more detail when we talk about *Git* & collaboration).

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

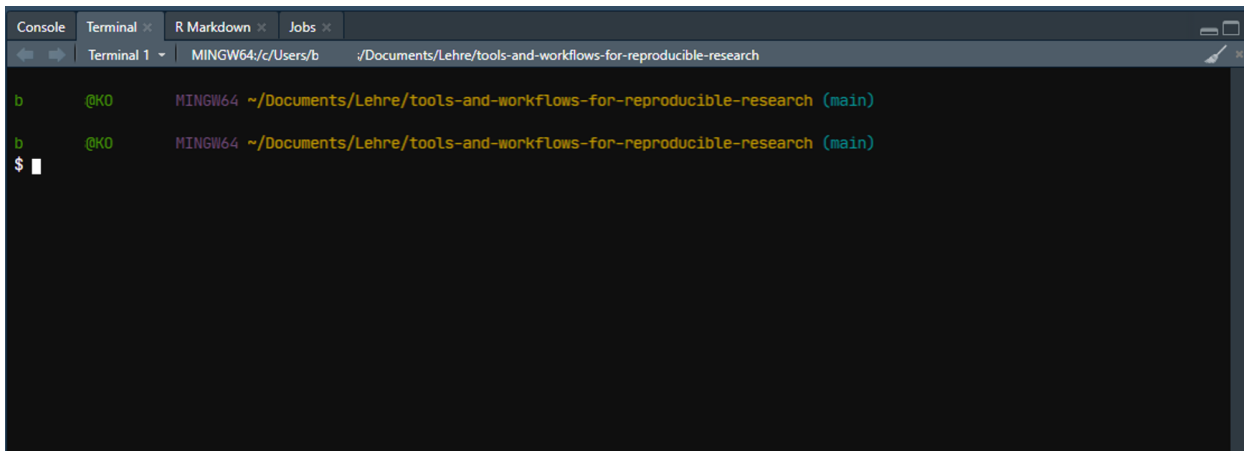
Lucky for us, if you need a CLI for using `Git`, you don't need to leave *RStudio*. As of version 1.3.1056-1, *RStudio* provides a `Terminal` tab in the console pane. Through this, *RStudio* provides access to the `system shell`. If you have properly installed `Git` you can use this to execute the full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



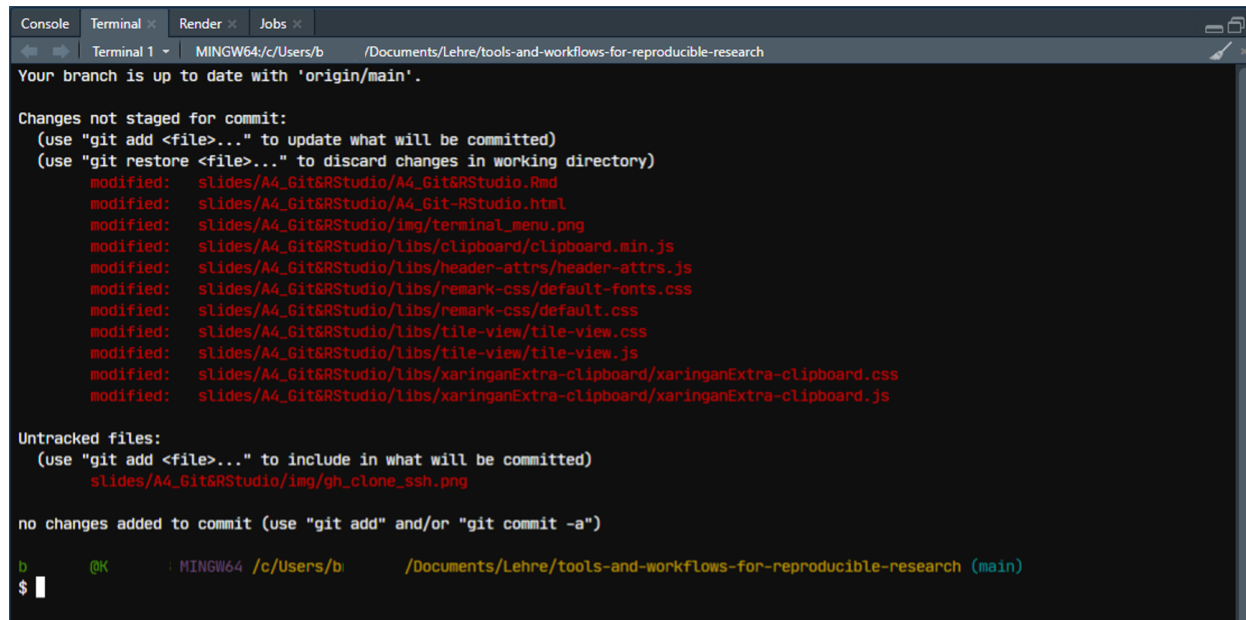
```
Terminal 1 | MINGW64/c/Users/b | /Documents/Lehre/tools-and-workflows-for-reproducible-research  
b @KO MINGW64 ~/Documents/Lehre/tools-and-workflows-for-reproducible-research (main)  
b @KO MINGW64 ~/Documents/Lehre/tools-and-workflows-for-reproducible-research (main)  
$
```

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 x Render x Jobs x
Terminal 1 MINGW64/c/Users/b /Documents/Lehre/tools-and-workflows-for-reproducible-research
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:   slides/A4_Git&RStudio/A4_Git&RStudio.Rmd
   modified:   slides/A4_Git&RStudio/A4_Git-RStudio.html
   modified:   slides/A4_Git&RStudio/img/terminal_menu.png
   modified:   slides/A4_Git&RStudio/libs/clipboard/clipboard.min.js
   modified:   slides/A4_Git&RStudio/libs/header-attrs/header-attrs.js
   modified:   slides/A4_Git&RStudio/libs/remark-css/default-fonts.css
   modified:   slides/A4_Git&RStudio/libs/remark-css/default.css
   modified:   slides/A4_Git&RStudio/libs/tile-view/tile-view.css
   modified:   slides/A4_Git&RStudio/libs/tile-view/tile-view.js
   modified:   slides/A4_Git&RStudio/libs/xaringanExtra-clipboard/xaringanExtra-clipboard.css
   modified:   slides/A4_Git&RStudio/libs/xaringanExtra-clipboard/xaringanExtra-clipboard.js

Untracked files:
  (use "git add <file>..." to include in what will be committed)
   slides/A4_Git&RStudio/img/gh_clone_ssh.png

no changes added to commit (use "git add" and/or "git commit -a")

b @K MINGW64 /c/Users/b /Documents/Lehre/tools-and-workflows-for-reproducible-research (main)
$
```


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"`

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* How-To Article on *Version Control with Git and SVN*.