gesis

Leibniz Institute
for the Social Sciences

Workflows for Reproducible Research with R & Git

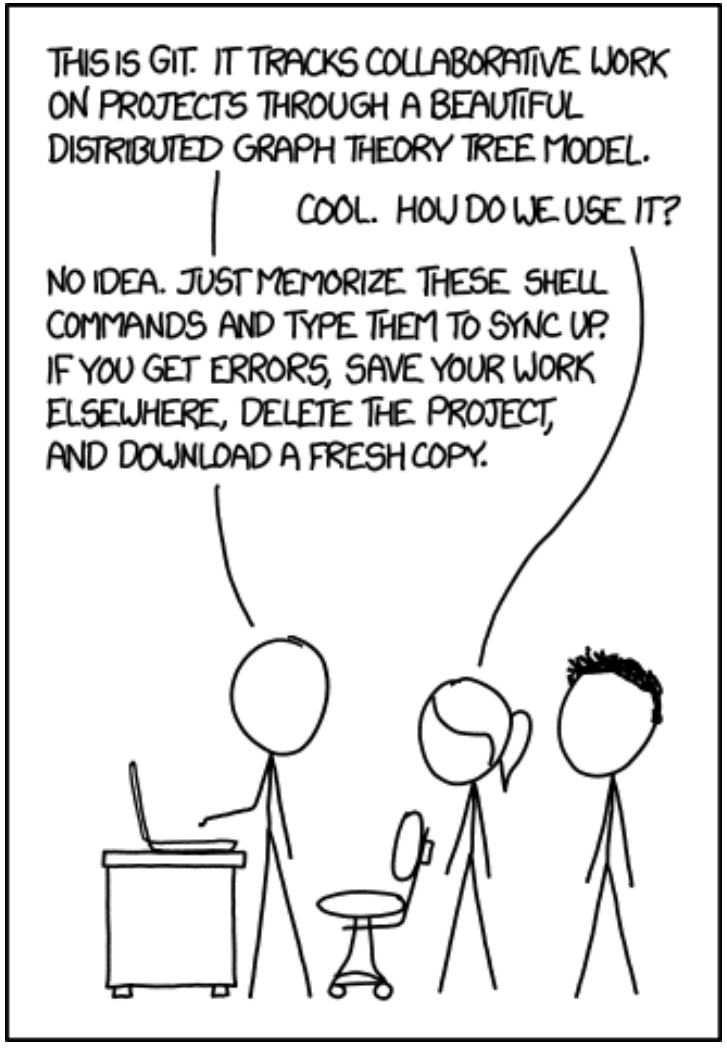
An introduction to Git

Bernd Weiß

2023-11-16

Leibniz Association

Git Part 1 -- An introduction



(Source: xkcd, https://xkcd.com/1597/, accessed on 2017-12-23)

Overview

Overview

- In part I, I will be introducing Git and GitHub
- Part II, we will introduce RStudio as a GUI to Git and talk more about GitHub and collaboration

The concept of version control

Preliminaries

- Git and other tools have been developed in the context of software development (in the Linux community, to be more precise)
- Even though there exits graphical user interfaces (GUIs) for working with Git, it is highly recommended that you have a basic knowledge of how to use the CLI
- Once you mastered using Git at the command line, using a GUI is a peace of cake

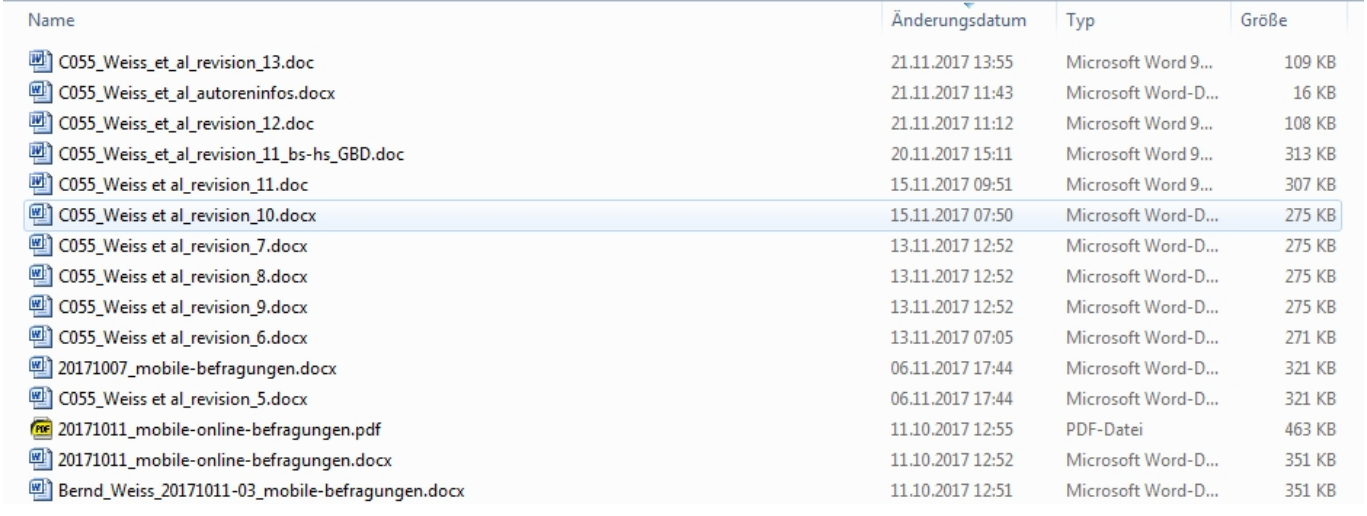
Why use a Version Control System?

- Backup
- Collaborative work and syncing
- Keeping track of changes and having a definitive "most recent" version of a file (see also next slide)
- Test new code/features in a "sandbox" (aka a new "branch")
- Log file of all changes (like a lab notebook)
- Authorship attribution
- ...

The horror of…

… `final_rev2_update12_after-computer-crashed.docx`

See also http://phdcomics.com/comics.php?f=1531



Name	Änderungsdatum	Typ	Größe
C055_Weiss_et_al_revision_13.doc	21.11.2017 13:55	Microsoft Word 9...	109 KB
C055_Weiss_et_al_autoreninfos.docx	21.11.2017 11:43	Microsoft Word-D...	16 KB
C055_Weiss_et_al_revision_12.doc	21.11.2017 11:12	Microsoft Word 9...	108 KB
C055_Weiss_et_al_revision_11_bs-hs_GBD.doc	20.11.2017 15:11	Microsoft Word 9...	313 KB
C055_Weiss et al_revision_11.doc	15.11.2017 09:51	Microsoft Word 9...	307 KB
C055_Weiss et al_revision_10.docx	15.11.2017 07:50	Microsoft Word-D...	275 KB
C055_Weiss et al_revision_7.docx	13.11.2017 12:52	Microsoft Word-D...	275 KB
C055_Weiss et al_revision_8.docx	13.11.2017 12:52	Microsoft Word-D...	275 KB
C055_Weiss et al_revision_9.docx	13.11.2017 12:52	Microsoft Word-D...	275 KB
C055_Weiss et al_revision_6.docx	13.11.2017 07:05	Microsoft Word-D...	271 KB
20171007_mobile-befragungen.docx	06.11.2017 17:44	Microsoft Word-D...	321 KB
C055_Weiss et al_revision_5.docx	06.11.2017 17:44	Microsoft Word-D...	321 KB
20171011_mobile-online-befragungen.pdf	11.10.2017 12:55	PDF-Datei	463 KB
20171011_mobile-online-befragungen.docx	11.10.2017 12:52	Microsoft Word-D...	351 KB
Bernd_Weiss_20171011-03_mobile-befragungen.docx	11.10.2017 12:51	Microsoft Word-D...	351 KB

Collaboration

Modern web interfaces such as GitHub also allow for social interaction

- Strangers can send you pull requests to improve your code/document/...)
- You can follow other interesting people or projects
- You can "star" projects to show your appreciation
- You can "fork" other projects
- ...

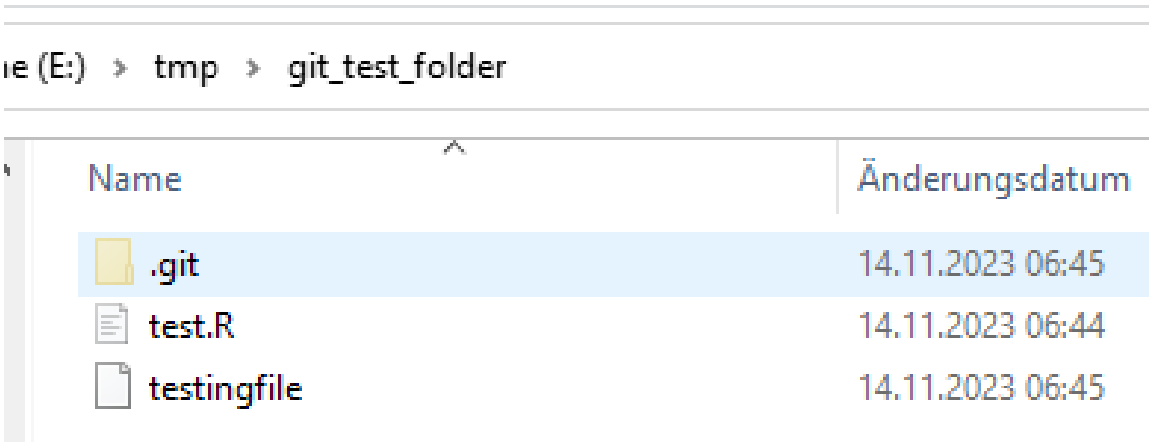
For what type of files is a VCS useful?

- Most useful for text files (Stata do files, SPSS syntax files, R skripts etc.). Text files can stored very efficient since only changes between version are tracked
- Binary files (Blob = binary large object) (images, MS Word files, Stata data files, etc.) can be stored in a VCS but less efficient than text files since every time the entire file is saved

Terminology and concepts

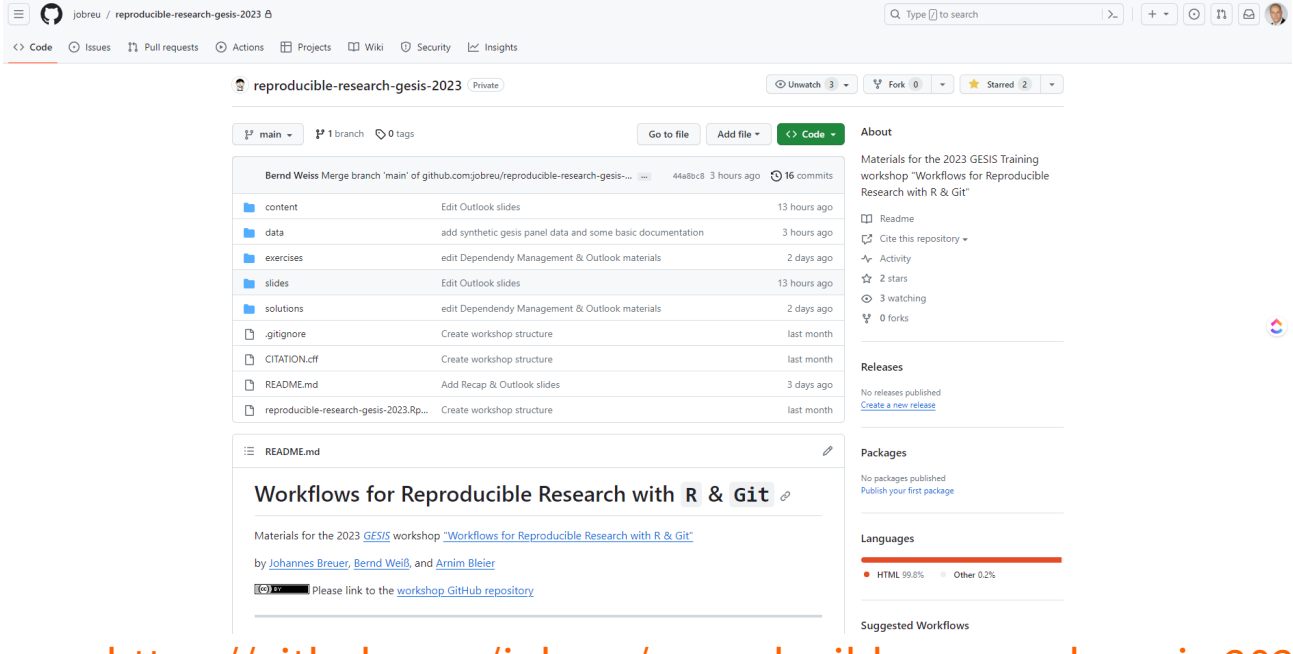
Git vs `git`

- There is Git.
- "Git" is the name of the software, and the actual command-line tool is `git` (e.g., in Windows it is `git.exe`).
- A (local) *Git repository* is a folder that is under "Git's version control"
- Locally, a Git repository always has a `.git` folder.



Git and collaboration

- GitHub, GitLab etc. are (web-based GUI) frontends, allowing to work collaboratively
- GitHub and GitLab also provide project management features



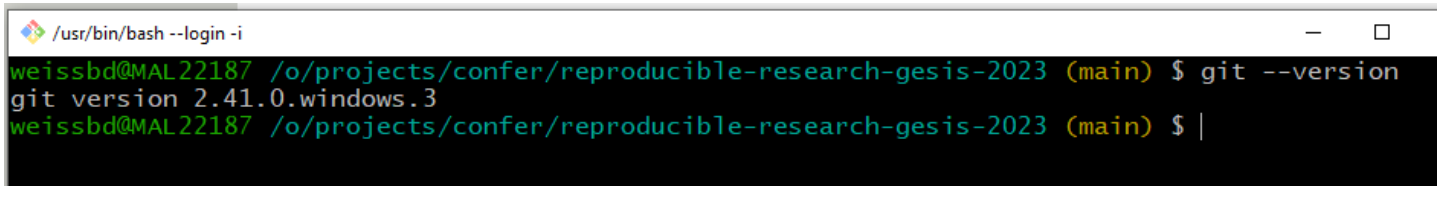
(Source: https://github.com/jobreu/reproducible-research-gesis-2023)

(!) Exercise: The Git Bash

Test your Git installation. Start the Git Bash and type in `git --version`:

```
git --version
```

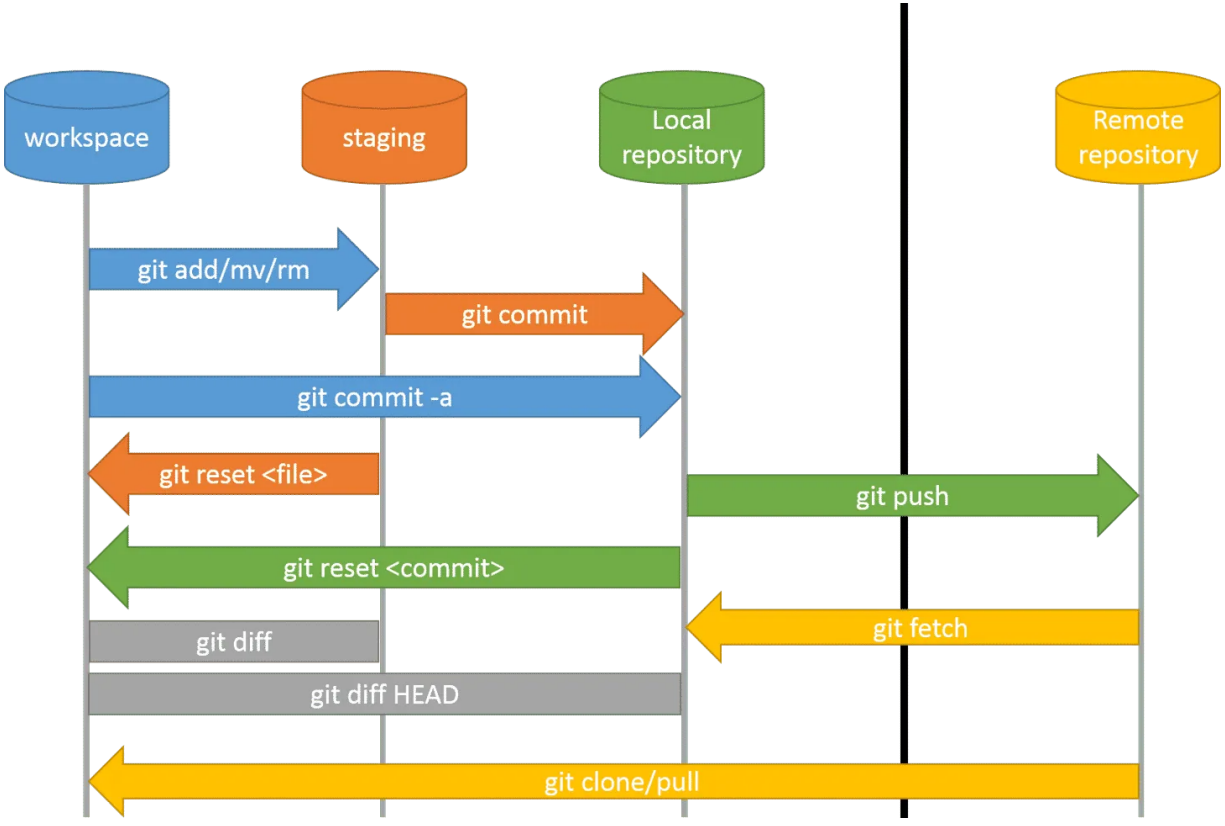
```
## git version 2.41.0.windows.3
```



(My) Workflow in Git

- Git is a very powerful tool, in my own work, I utilize a rather limited set of its capabilities
- Work locally (i.e., on your computer) on your files until a certain feature is completed (a function is completed, a paragraph written etc).
- `Commit` your file and write a commit message, i.e., provide inform that a certain file (or more) have changed and inform your future self (or someone else) about the nature of your changes (aka write a commit message). This has to be done manually.
- Commit early, commit often!
- When a remote repository exists: send (`push`) your changes to the remote repository.

Visualization of a Git workflow



(Source: https://teaching.dahahm.de/teaching/ss22/dissys/2022/05/31/git_workflow.html)

Why and how I use Git

- This is what I mostly do with Git:
  - Initialize a new Git repository or clone an existing respository
  - Backup my work on a remote server
  - Track changes
  - Use branches to implement experimental features
  - Search (and undo) previous changes (most of the time using the interface provided by GitHub or GitLab)
- "Google" (or whatever your prefered seach engine is) a lot ...

Git: A 30,000 foot view

- Git is a version control system (VCS). As mentioned above, a VCS allows you to track the history and attribution of your project files over time in a repository (Narębski, 2016)
- It is, if you will, a (very, very) powerful undo function (well, kind of...)
- To be more precise, Git is a distributed VCS (DCVS) and hence a tool for collaborative work
- If you want to utilize Git for collaborative work, one approach of using Git in this context assumes that there exists a central and remote repository. Most famous is GitHub, at GESIS we use GitLab

Installing Git and setup

Download and installation

Git (for Windows) can be downloaded from: https://git-scm.com/download/win.

Here are a few questions that you will be asked during the installation:

- Default editor (use Notepad++ if you have it on your computed, vim also works)
- Adjusting your PATH environment (you might want to go with the second option "Use Git from the Windows command Prompt")
- ...

Download and installation (cont.)

In case you will be working with others, you also will need a remote repository (be able to access a remote repository). For convenience reasons it is recommended that you also install/set up SSH (see next slides).

For various reasons, I no longer use a standalone version of Git but use a version of Git that can be installed via MSYS2.

> "MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software." -- https://www.msys2.org/.

Well, not so distributed at all...

- Even though Git is called "distributed", most of the time, there is just one central server (e.,g., GitHub, GitLab, ...)
- A Git project is stored in a repository, which can be local or remote
- When using Git to access a remote repository (for backup or collaborative work) on a remote server, you need to authenticate yourself to the server
- There are two ways of authentication: HTTPS or SSH

Authentication

- Despite its technical details, I tend to choose SSH, but Johannes, for instance, prefers HTTPS (in part II I will introduce the HTTPS approach)

- More information can be found on these websites:

  - https://happygitwithr.com/index.html
  - https://happygitwithr.com/https-pat.html
  - https://happygitwithr.com/ssh-keys.html
  - https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories
  - https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/about-authentication-to-github#authenticating-with-the-command-line

Basic workflow

Overview basic workflow

1. In the very beginning: Obtain a repository from a remote server (`git clone`) *or* initialize it yourself locally (`git init`) -- this is done only once!

2. Check files into your local repository (`git add` and `git commit`)

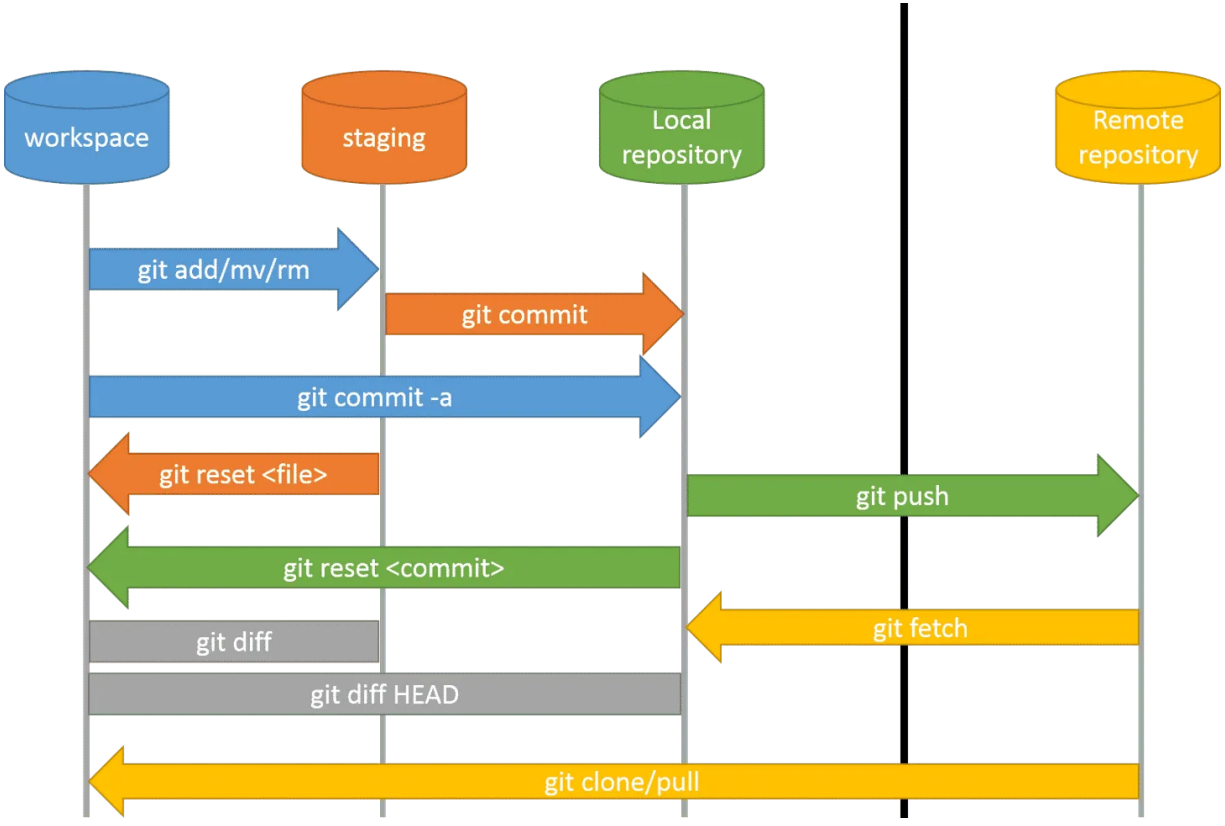
3. Do some work

4. See 2. or ...

5. ... send local work to remote server (`git push`)

6. In a collaborative setting and once step 1. has been completed, updates by your collaborators can be downloaded to your local repository (`git pull`)

Visualization of a Git workflow



(Source: https://teaching.dahahm.de/teaching/ss22/dissys/2022/05/31/git_workflow.html)

Setting up a Git repository

Usually, there are two ways to set up/obtain a Git repository:

1. You create a new Git repository or ...
2. ... you "clone" an existing repository from a remote Git server such as GitHub/GitLab

The sample R file

I will be working with the following example file `test.R`:

```
## l1: # Branch: main
## l2: # Author: BW
## l3: # Always start with a dumb comment
## l4: x <- c(1:10)
## l5: mean(x)
## l6: var(x)
## l7: sum(x)
```

Step 1: Creating a local Git repository

The first step is to create a Git repository. After the repository has been created, we need to tell `git` which files will be subject to version control. So, the following git commands will be utilized:

- `git init`: Creates a new folder `.git`, which contains configuration files and the repository. As of now, `git` does not know anything about our file(s), e.g., `test.R`

Content of `git_test_folder` before `git init`

From now on, all examples will refer to a demo repository called `git_test_folder`

```
## total 5
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 ..
## -rw-r--r-- 1 weissbd GESIS+Group(513) 131 Nov 16 07:41 test.R
```

And, again, `test.R` contains the following content:

```
## l1: # Branch: main
## l2: # Author: BW
## l3: # Always start with a dumb comment
## l4: x <- c(1:10)
## l5: mean(x)
## l6: var(x)
## l7: sum(x)
```

Initialize a Git repository

Now, let's initialize the Git repository using the `git init` command

```
cd e:/tmp/git_test_folder
git init
```

```
## Initialized empty Git repository in E:/tmp/git_test_folder/.git/
```

```
cd e:/tmp/git_test_folder
ls -la
```

```
## total 9
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .git
## -rw-r--r-- 1 weissbd GESIS+Group(513) 131 Nov 16 07:41 test.R
```

git status

Let's check the status of our newly created repository using the command `git status`. It shows the status of the current working tree (and branch). As of now, `git` is not aware of any files yet, so it informs us about the existence of 'Untracked files: ...'.

```
cd e:/tmp/git_test_folder
git status
```

```
## On branch main
##
## No commits yet
##
## Untracked files:
##   (use "git add <file>..." to include in what will be committed)
##     test.R
##
## nothing added to commit but untracked files present (use "git add" to track)
```

(Note: the main branch is called `main`; it used to be called "master", the new convention, though, is "main")

Step 2: Adding files to a git repository: `git add` and `git commit`

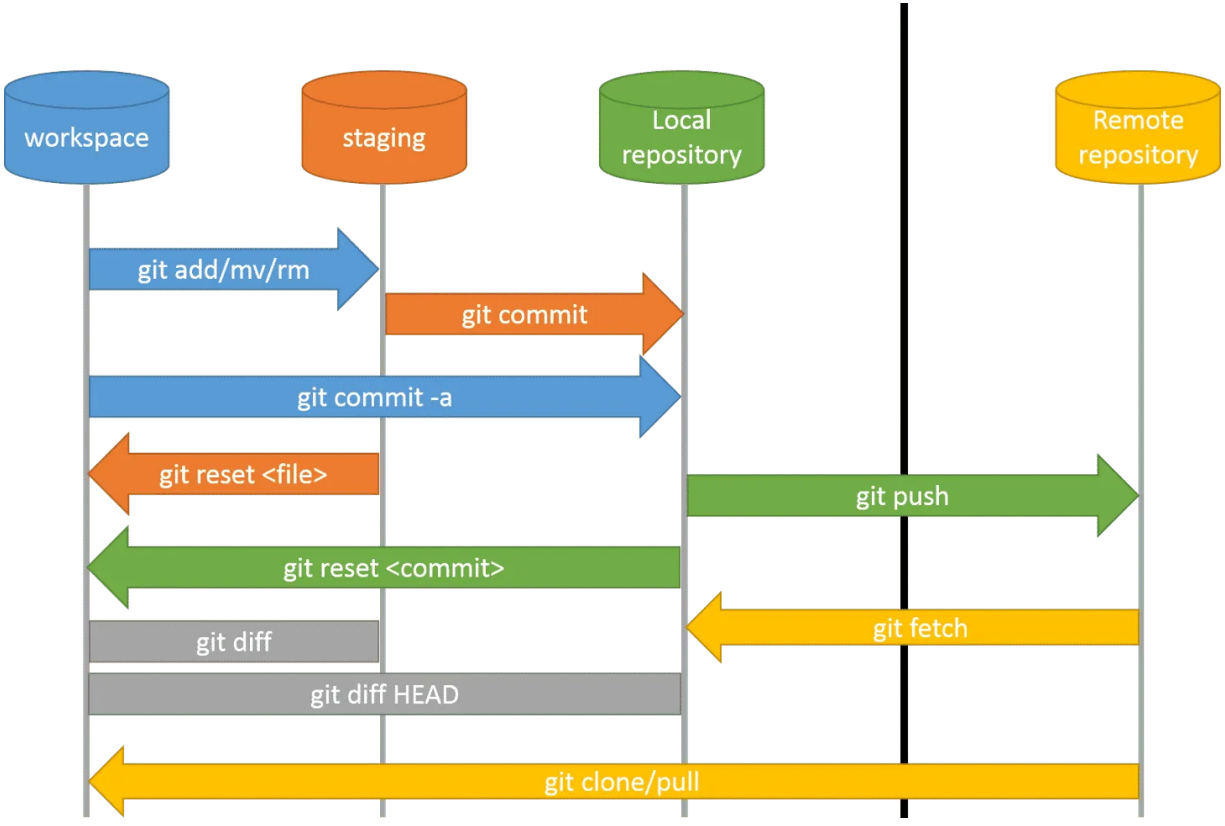
Now it is time for some file action by adding (a) file(s) to our repository.

In the previous section on `git status`, it was recommended that `git add` is used to add files to the git repository:

```
(use "git add <file>..." to include in what will be committed)
```

That is what we are going to do now: To actually 'save' (check-in or track) files in the repository, a *two-step* procedure needs to be performed.

Visualization of a Git workflow



(Source: https://teaching.dahahm.de/teaching/ss22/dissys/2022/05/31/git_workflow.html)

Getting files into your local repository: `git add`

The first step is to call `git add`, the second step is to commit the file(s) using `git commit`.

For now, it might be hard to see the benefit of this two-step procedure, see http://gitolite.com/uses-of-index.html for a thorough description (I like the "staging helps you split up one large change into multiple commits" argument).

- `git add -A` : Adds (here `-A` means "all files") files to the *index* (or staging area) -- note: this approach is against the principle of "focus on one aspect of the code changes".
- To add a particular files to the index, use `git add my_special_file.do`.

Getting files into your local repository: `git add`

Run `git add`:

```
cd e:/tmp/git_test_folder
git add -A
```

Let's see what `git status` has to say; the "untracked files" are gone

```
cd e:/tmp/git_test_folder
git status
```

```
## On branch main
##
## No commits yet
##
## Changes to be committed:
##   (use "git rm --cached <file>..." to unstage)
##     new file:   test.R
```

Getting files into your... : `git commit`

The second step is to run the command

`git commit -m "your text, verbs in imperative form"` (see below), e.g.

`git commit -m "add function to compute tau^2".`

Since this is my first commit, I always apply the following commit message: `git commit -m "initial commit.`

Getting files into your... : `git commit`

```
cd e:/tmp/git_test_folder
git commit -m "Initial commit"
```

```
## [main (root-commit) 27b2b6c] Initial commit
##  1 file changed, 7 insertions(+)
##  create mode 100644 test.R
```

According to the Git developer site commit messages should follow the "imperative-style":

> "Describe your changes in imperative mood, e.g. "make xyzzy do frotz" instead of "[This patch] makes xyzzy do frotz" or "[I] changed xyzzy to do frotz", as if you are giving orders to the codebase to change its behavior. Try to make sure your explanation can be understood without external resources. Instead of giving a URL to a mailing list archive, summarize the relevant points of the discussion."

Getting files into your... : `git commit`

Again, let's see what `git status` reports:

```
cd /e/tmp/git_test_folder
git status
```

```
## On branch main
## nothing to commit, working tree clean
```

So, there are no untracked files, "nothing to commit, working directory clean".

Git's commit history: `git log`

There is another useful command `git log` that informs about `git`'s history (like a lab notebook), i.e. commited files and folders:

```
cd e:/tmp/git_test_folder
git log
```

```
## commit 27b2b6c09e1937e7dbb957d54ed2ae628f930c99
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:20 2023 +0100
##
##     Initial commit
```

Right now, the history only contains one entry.

The very first line `commit ...` shows the SHA1 hash. The 'Secure Hash Algorithm 1' is used to calculate this long, hexadecimal number for a file. Files with identical content are represented by an identical SHA1 hash, files with different content do not share an identical SHA1 hash. Using these SHA1 numbers, `git` can identify changes in a file.

Security -- Some things do NOT belong in your Git repo

- Be extremely careful including sensitive information (e.g., personal data, passwords, access tokens) into a (public) GitHub repository. There are people out there who search for these things... see also https://docs.github.com/en/code-security/secret-scanning
- Use a `.gitignore` file to exclude sensitive files/folders
- The text file `.gitignore` is placed directly in your repo's root (see next slide)
- Here is an example:

```
## analyses-freda-ws_cache/
## data/
## *.RData
## .Rproj.user
```

- For more information, see https://git-scm.com/docs/gitignore

More security

- Please enable two-factor authentication on GitHub
- In case you have accidentally include sensitive information, check out this GitHub website on Removing sensitive data from a repository

Location of `.gitignore` in your repo's root

```
ls -la
```

```
## total 52
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 16 07:11 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 16 06:53 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 15 14:24 .Rproj.user
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 16 07:11 .git
## -rw-r--r-- 1 weissbd GESIS+Group(513)  570 Nov 15 14:24 .gitignore
## -rw-r--r-- 1 weissbd GESIS+Group(513)  539 Nov 15 14:24 CITATION.cff
## -rw-r--r-- 1 weissbd GESIS+Group(513) 6833 Nov 16 04:55 README.md
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 15 14:24 content
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 15 14:24 data
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 15 14:24 exercises
## -rw-r--r-- 1 weissbd GESIS+Group(513)  218 Nov 16 07:11 reproducible-research-gesis-2023.R
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 16 04:55 slides
## drwxr-xr-x 1 weissbd GESIS+Group(513)    0 Nov 15 14:24 solutions
```

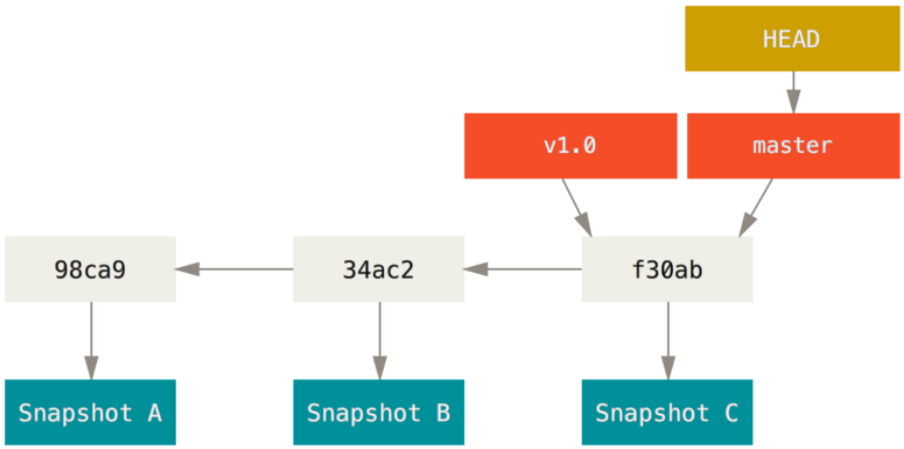
(!) Exercise

1. Open your Git Bash
2. Go to your Home directory via `cd ~` (or, actually, go wherever you want)
3. Create a new folder (e.g., via `mkdir` or `create-project.sh`, see https://github.com/jobreu/reproducible-research-gesis-2023/tree/main/content/sh)
4. Change into the newly created directory via `<your input here>`
5. Initialize your new Git project via `git init`
6. Copy a few files (PDF files etc. -- does not really matter, but no sensitive material!) in your new project folder
7. What comes next? Hint: `git add` and then `git commit <your input>`
8. Check the status and the history of your Git repository

More terminology

The magic of DAGs (HEAD, commit IDs, ...)

- From a user perspective, important "building blocks" of a repository are commits (and branches).
- Git's history (the sequence of commits) is based on a directed acyclic graphs (DAG).



(Source: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell)

More information on Git and DAGs

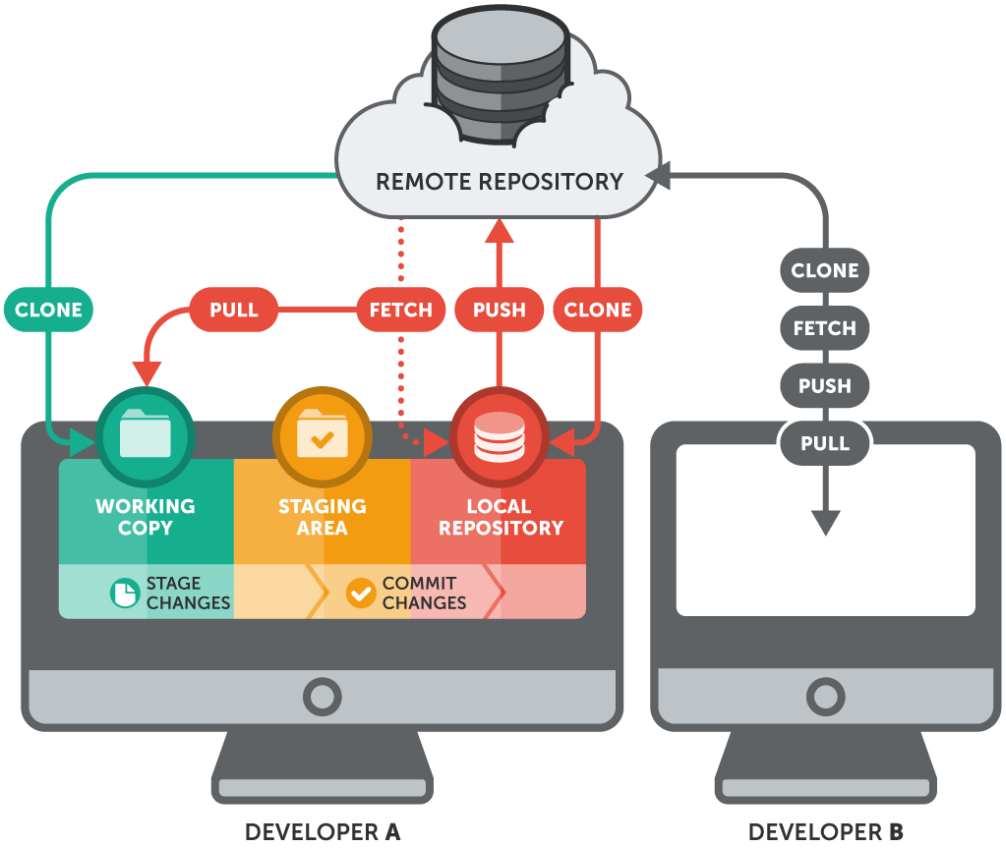
This is just a brief excursus and I am including the link to an example of a Git DAG:

https://subscription.packtpub.com/book/application-development/9781782168454/1/ch01lvl1sec11/viewing-the-dag

GitHub

Disclaimer

Some of the following slides are heavily influenced by another course on "Reproducible research workflows for psychologists" by Frederik Aust and Johannes Breuer, especially the part on "Collaborate with Git & GitHub"



(Source: https://www.git-tower.com/learn/git/ebook/en/desktop-gui/remote-repositories/introduction)

Working with remote repositories

- As mentioned in the introduction, Git is especially powerful when it comes to collaborative work, e.g., via GitGub or GitLab.
- In order to work with others, you need some sort of connection to these other person(s). The one I am discussing here is having a central remote repository C.
- Let us assume that you have another collaborator. Then you as well as the other person need to synchronize with the same repository C.
- There also exists another model which is based on a decentralized approach, where you could individually sync with x-y, x-z, y-z etc.

Establishing a connection to a remote repository

There are two ways to establish a connection to a remote repository (e.g., on GitHub):

1. Clone a remote repository via `git clone ...`.
2. Setting up a new remote repository via `git remote add <name> <url>`.

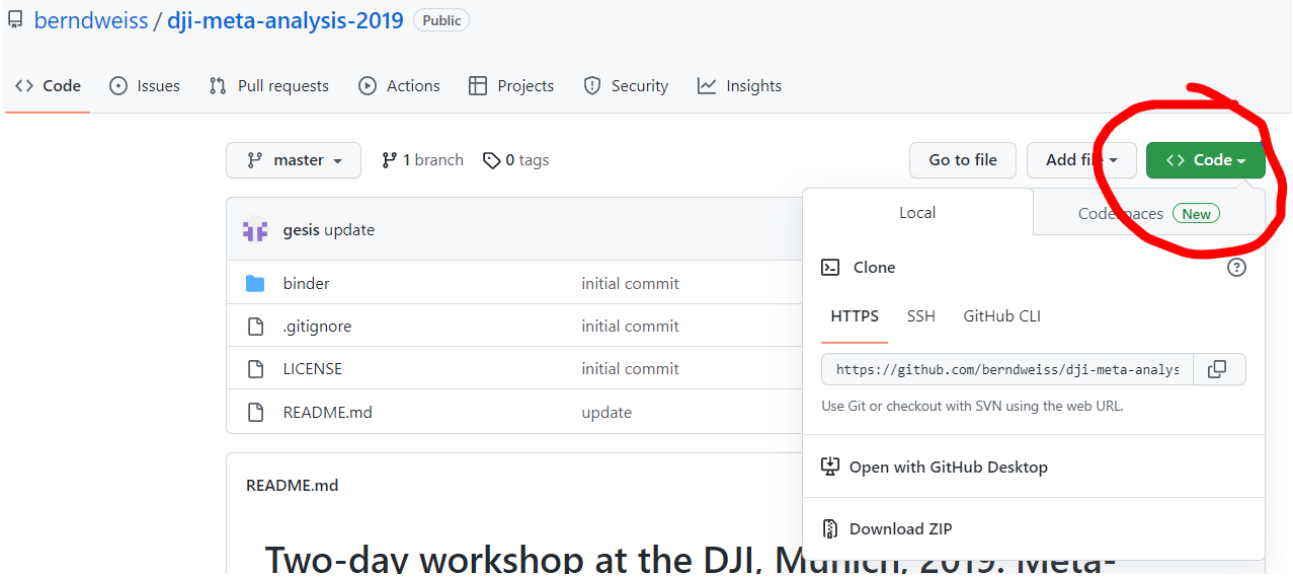
Cloning

Cloning a remote repository

- Cloning a remote repository via GitHub/GitLab/... is quite easy
- Visit the website, on GitHub look for the green "Code" button, see also the screenshot below
- Decide whether you would like to use the HTTPS or SSH protocoll
- Copy the link and execute `git clone`
- Cloning -- in contrast to "Download ZIP" -- means that you also download the entire Git history (i.e., the `.git` folder)
- Note: Of course, you need to have access to the GitHub repository, i.e., it is either a public repo or you have been granted access

How to clone a (public) remote repository I

- Here is an example using my workshop on "Meta-Analysis in Social Research" (public repo), see https://github.com/berndweiss/dji-meta-analysis-2019



How to clone a remote repository II

Open a CLI and execute: `git clone https://github.com/berndweiss/dji-meta-analysis-2019.git`

```
cd e:/tmp
git clone https://github.com/berndweiss/dji-meta-analysis-2019.git
```

```
## Cloning into 'dji-meta-analysis-2019'...
```

```
cd e:/tmp
ls -la
```

```
## total 21
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 06:34 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov  9 18:53 20231109_freda-workshop
## -rwxr-xr-x 1 weissbd GESIS+Group(513) 195 Nov 15 18:32 create-project.sh
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 dji-meta-analysis-2019
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 git_test_folder
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 14 08:03 lala
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Sep 14 15:52 meta_k12
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 14 19:09 newproj
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 14 18:37 ps2021-10-ws-repro-research_bw-slid
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 15 17:10 renv-sample-project
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 15 18:33 testfolder
```

Authentication

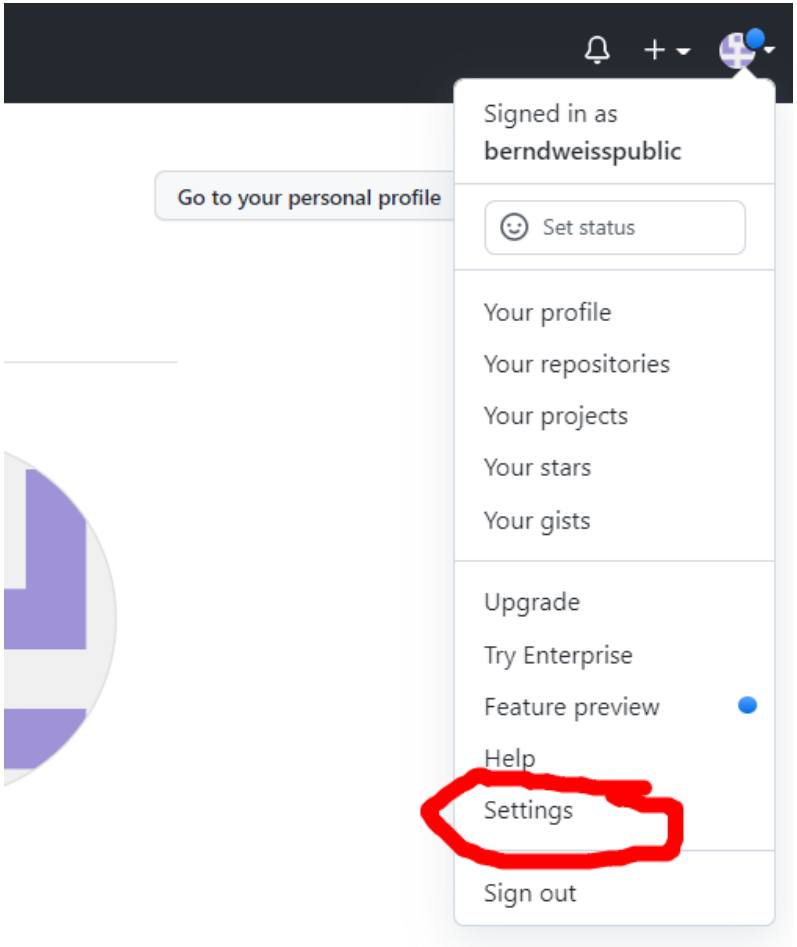
Authentication: Local or remote & HTTPS or SSH?

- A Git project/repo is stored in a repository, which can be local or remote
- When using Git to access a (nonpublic, aka private) remote repository (for backup or collaborative work) on a remote server, you need to authenticate yourself to the server
- There are two ways of authentication: HTTPS or SSH

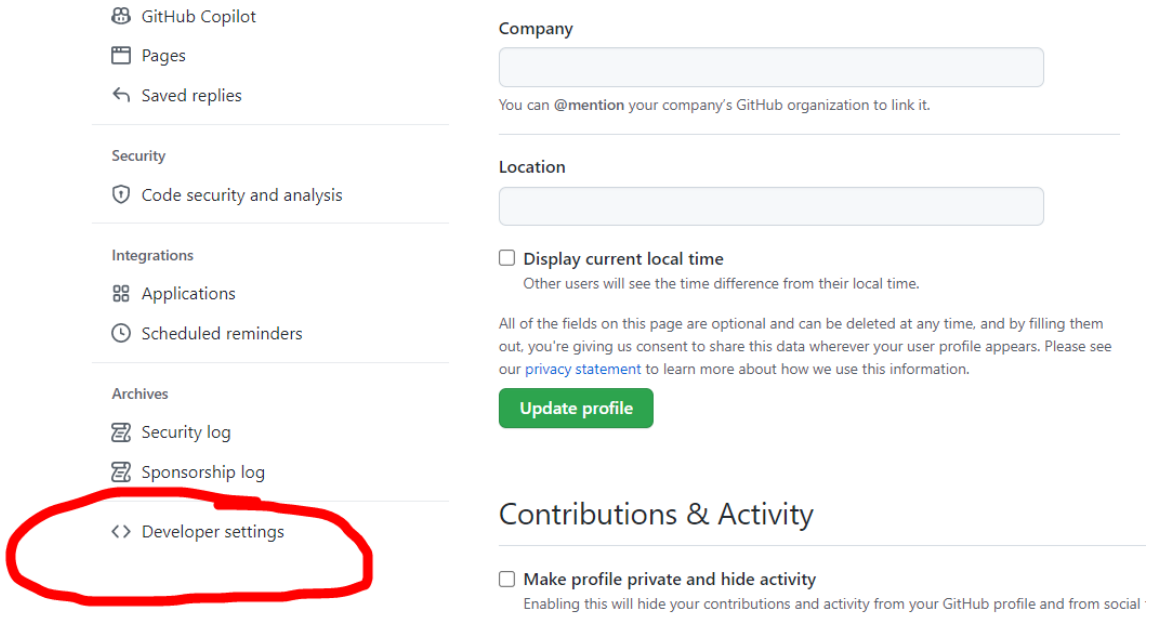
GitHub: Using personal access tokens

- These days, authentication via personal access tokens (PAT) (and https) seems the way to go when using GitHub
- In the following, I will illustrate the process using multiple screenshots
- Note that my explanation does not include any R/RStudio-related processes. Johannes talks about these things in more detail
- Finally, I will focus on MS Windows. Arnim can help with MacOS/Linux.

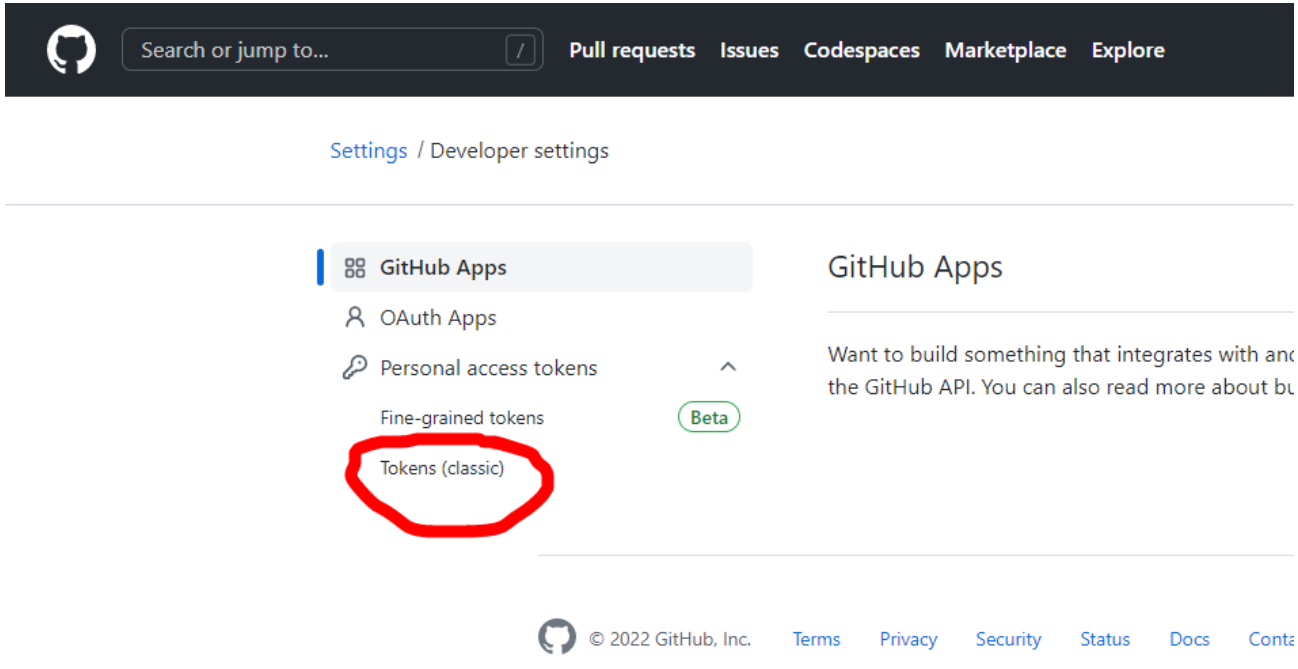
In GitHub, go to the `Settings` website:



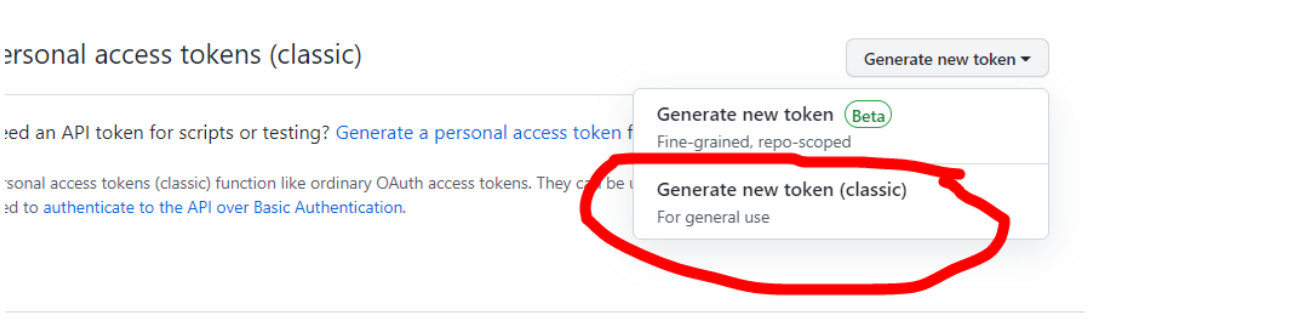
Next, go to the `Developer Settings` entry:



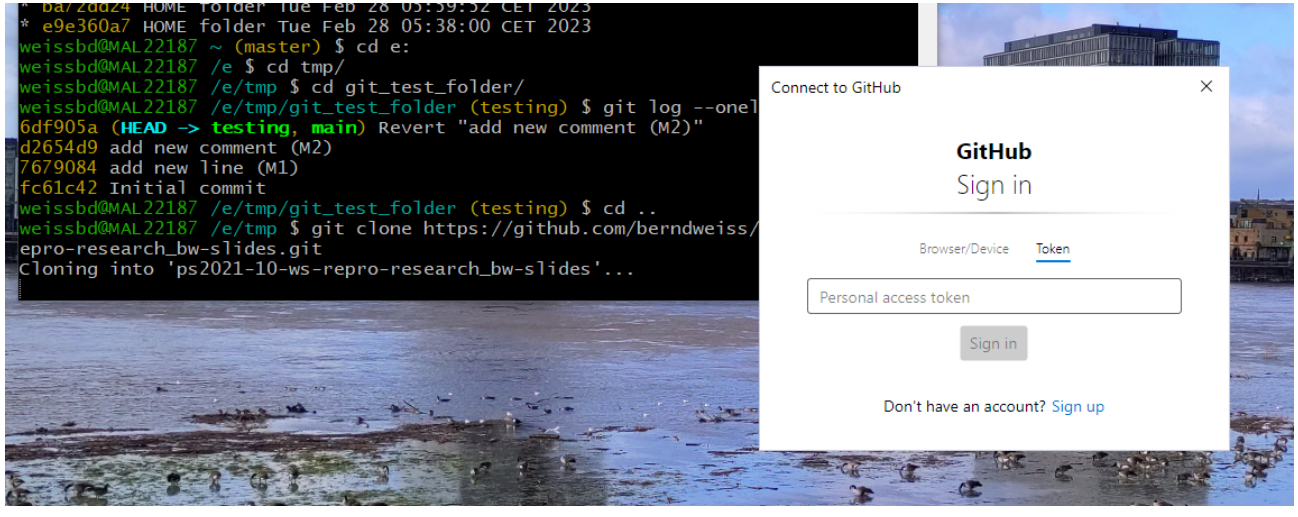
Then, choose `Tokens (classic)`:



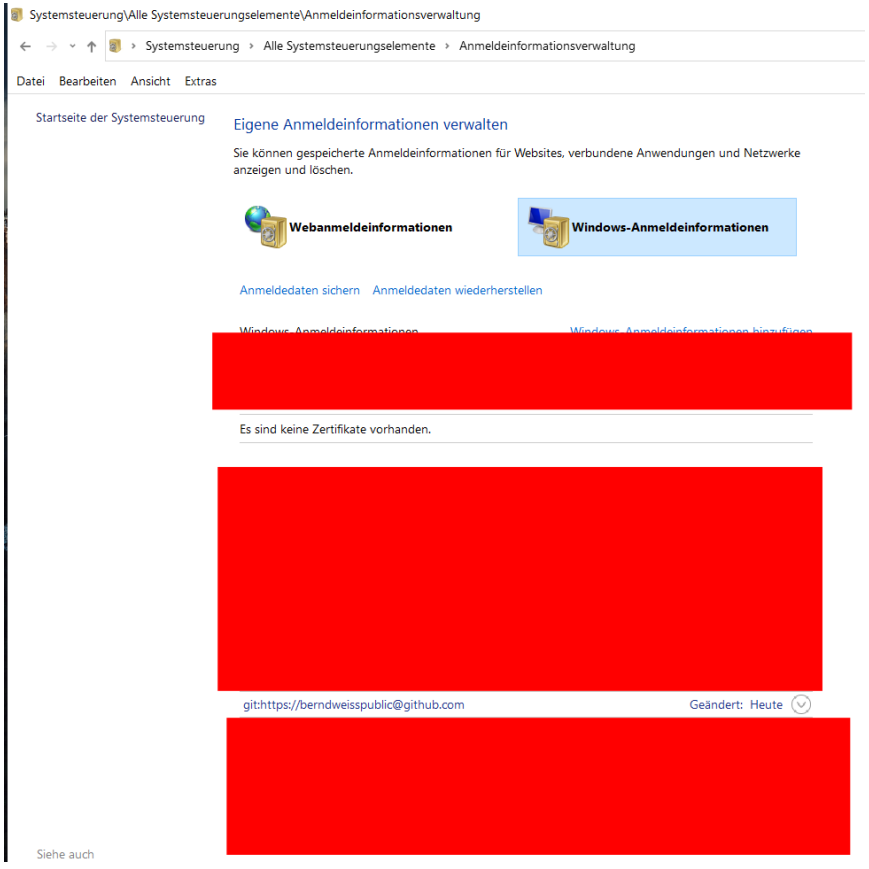
And, generate a new token; important, save this Token (e.g., in your password manager):



When you are now cloning a new repository (or pushing/pulling **for the first** time), you will be asked once to enter your Token



If, for whatever reason, you decide to reset/remove your credentials, you can do so using the Windows Credentials Manager (in German: "Anmeldeinformationsverwaltung")



Setting up SSH

- An alternative to using a PAT, is using SSH.
- SSH is a network protocol that comes in handy, when you work with remote repositories and when you do not want to type-in your password every time you pull (fetch) or push (send) from a remote repository. You still need to authenticate yourself, though.
- To work with Git on you local computer, you do not need SSH (= Secure Shell).
- Authentication in SSH (which is also the name of the program) works by using a private and a public key (usually the public key has the file extension `.pub`, e.g., my public key is `id_rsa.pub`). When you start working with SSH for the very first time, you have to create both keys.

- The private key remains on your local computer and you have to make sure that it is safe -- it is a simple text file and it is your password now, and everyone who has your private key can access your files. Again, everyone who has your private key has your password!

This is what my *public key* looks like:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAyOQ9RT6TkfgkdO2NspzdVJE5CZ03yYAhVwLGo
CrI3E9/Ix0MAySunXExjhsQi2XkhPBjLOEahYuuLaAWHuBc7apUPRNSBy+mdUHnH3
0BdTQijQ6vj3RL99HO4yrZnipIlkS5ufw/+hpbXXOzSOqTvyGtL9ygm3eA2HDSQtz
2ptFq8anODJDKrgTbNLb/YZ9KDIcpdO/Sfk4LtvaGF3tIFlyE+pogNmN4eWiYg9Xv
25BhVVxWMHadRFLeDastWO4SedriEHzQYaNgxVNTufqolJ0nbg4R//fVDxjR2SbzV
AHLZ+eVPUx+vzcPVMP9wYPcnii9YLiSRy+hlUAOR/kXeQ== berndweiss
```

The *public key* (not the private key!) has to be stored at the GitHub/GitLab/... website. Now, everyone who has your public key can encrypt files (that are sent to you via the internet) but only you (or anyone else who has your private key) can decrypt the files. And, for that reasons you do not have to login everytime you push/pull files from the remote repository.

- How to setup SSH on you computer is explained on this website: https://docs.gitlab.com/ce/ssh/README.html ("Generate an SSH key pair")

- The most important point is that `ssh` is able to find your key pair, i.e., it needs to be located in your HOME folder

If everything works well, you should receive the following friendly welcome message after typing in the command `ssh -T git@github.com`:

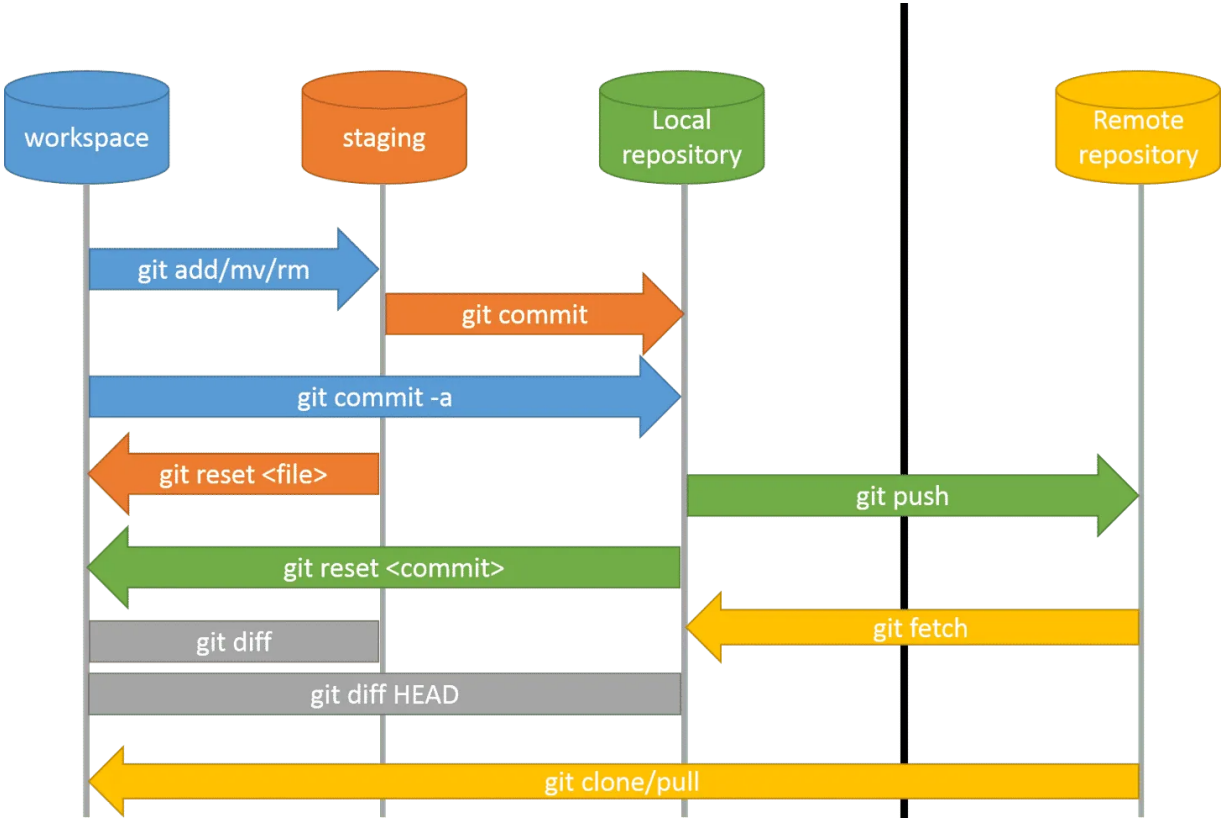
> Hi berndweiss! You've successfully authenticated, but GitHub does not provide shell access.

Set up a new remote respositories

Adding a new remote repository via `git remote add`

- Adding a new remote repository can be accomplished using the git command `git remote add <name> <url>`. The usual name for `<name>` is `origin`, however, feel free to choose another name (when you clone a repo, then this has already happened).
- SSH: The `<url>` for this repository looks like `git@git.gesis.org:weissbd/ps2017-xx-intro2git.git`; another example is this one: `git@github.com:berndweiss/ps2017-11_porto-campbell-ma-workshop.git`.
- HTTPS: `git remote add origin https://github.com/berndweiss/lala.git`

Visualization of a Git workflow

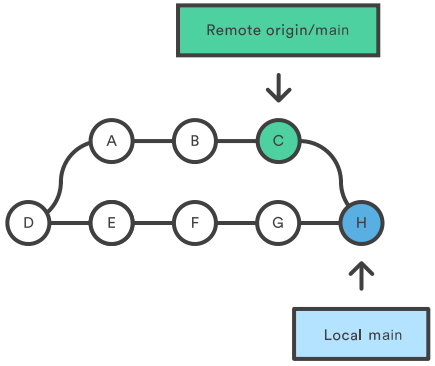


(Source: https://teaching.dahahm.de/teaching/ss22/dissys/2022/05/31/git_workflow.html)

`git pull`

- Given that you have already established a link to an remote server (such as GitHub, e.g., via `git remote add` or `git clone`), updates can be downloaded via the `git pull <name remote server> <branch>` command
- Most often, this is: `git pull origin main`
  - `origin` is an arbitrary name
  - `main` is the respective branch

- In the background, `git pull` combines two steps, `git fetch` and `git merge`



(Source: https://www.atlassian.com/git/tutorials/syncing/git-pull)

`git push`

- Again, given that you have already established a link to an remote server (such as GitHub, e.g., via `git remote add` or `git clone`), updates can be uploaded via the `git push <name remote server> <branch>` command
- Most often, this is `git push origin main`
- More information can be found here: https://www.atlassian.com/git/tutorials/syncing/git-push

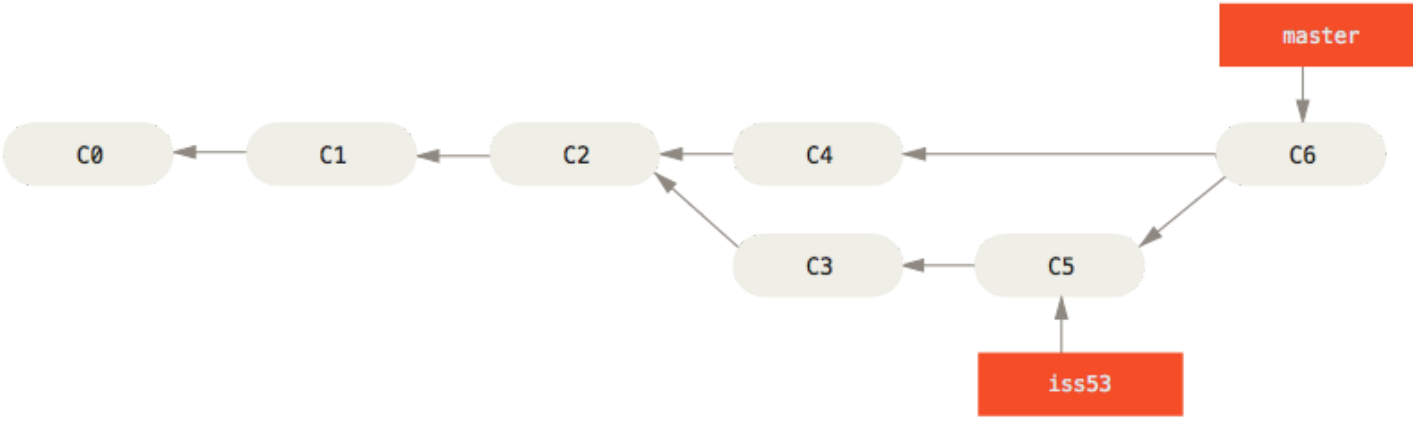
Git Part 2 -- The (slightly more) advanced stuff

Local branches

Branching (local)

- In addition to providing a powerful undo function, Git also allows to "toy around" with different (parallel) "versions" of your text or code
- Let's assume that you wrote a first draft of an R script. Everything works as expected. From a programming perspective, though, the script is just ugly and it is therefore quite hard to add additional features
- What I used to do was: save my original file as `my-great-program.R` and start working on a new version of the program using a file called `my-great-program_new.R`
- This is not necessary with `git branch`

Visualization of branches in Git



(Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging)

Example of a (local) branch I

Let's start with a list of files that are currently in my project folder:

```
cd e:/tmp/git_test_folder
ls -la
```

```
## total 9
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .git
## -rw-r--r-- 1 weissbd GESIS+Group(513) 131 Nov 16 07:41 test.R
```

```
cd e:/tmp/git_test_folder
git status
```

```
## On branch main
## nothing to commit, working tree clean
```

Example of a (local) branch II

What branches are available? Once we have more than one branch, the asterisk `*` shows which branch is active (or: in which branch we are in)

```
cd e:/tmp/git_test_folder
git branch
```

```
## * main
```

Create a new branch called `testing`

```
cd e:/tmp/git_test_folder
git branch testing
```

```
cd e:/tmp/git_test_folder
git branch
```

```
## * main
##   testing
```

Example of a (local) branch III

How do we get into the `testing` branch? Use `git checkout testing`

```
cd e:/tmp/git_test_folder
git checkout testing
git branch
```

```
## Switched to branch 'testing'
##   main
## * testing
```

Example of a (local) branch IV

Create a new file `testingfile`

```
cd e:/tmp/git_test_folder
touch testingfile
echo "in testing" > testingfile
ls -la
```

```
## total 10
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .git
## -rw-r--r-- 1 weissbd GESIS+Group(513) 131 Nov 16 07:41 test.R
## -rw-r--r-- 1 weissbd GESIS+Group(513)  11 Nov 16 07:41 testingfile
```

Example of a (local) branch V

```
cd e:/tmp/git_test_folder
cat testingfile
```

```
## in testing
```

```
cd e:/tmp/git_test_folder
git add testingfile
git commit -m "new branch testing"
```

```
## [testing 520dd2b] new branch testing
##  1 file changed, 1 insertion(+)
##  create mode 100644 testingfile
```

Example of a (local) branch VI

Switch back to branch `main` (and `cat testingfile` should result in an error message, since there is no `testingfile` in branch `main`)

```
cd e:/tmp/git_test_folder
git checkout main
ls -la
```

```
## Switched to branch 'main'
## total 9
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .git
## -rw-r--r-- 1 weissbd GESIS+Group(513) 131 Nov 16 07:41 test.R
```

Example of a (local) branch VII

Now, we can use `merge` to combine `main` and `testing`

```
cd e:/tmp/git_test_folder
git merge testing
```

```
## Updating 27b2b6c..520dd2b
## Fast-forward
##  testingfile | 1 +
##  1 file changed, 1 insertion(+)
##  create mode 100644 testingfile
```

```
## total 10
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 ..
## drwxr-xr-x 1 weissbd GESIS+Group(513)   0 Nov 16 07:41 .git
## -rw-r--r-- 1 weissbd GESIS+Group(513) 131 Nov 16 07:41 test.R
## -rw-r--r-- 1 weissbd GESIS+Group(513)  11 Nov 16 07:41 testingfile
```

```
cd e:/tmp/git_test_folder
cat testingfile
```

```
## in testing
```

Moving back in time

Undo changes

- Undoing changes can be done utilizing three different approaches (`git checkout`, `git revert`, `git reset`)
- **Depends on the state of your working directory (clean or uncommitted changes), publication status, your willingness to change Git's history etc.**
- A pragmatic approach is to utilize the search functionality of a web platform such as GitHub or GitLab
- Here, only some basics will be introduced, further information is provided by https://www.atlassian.com/git/tutorials/undoing-changes, https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting or https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things
- Nice flowchart: http://justinhileman.info/article/git-pretty/git-pretty.png

git checkout

- In order to undo (a) *uncommitted* changes or (b) going back to an earlier commit, respectively, the command `git checkout` can be utilized
- You have multiple possibilities to undo changes. You can undo changes regarding a particular file or you can go back to an earlier brench or commit[1], which may contain multiple changes (not a good practice, though)
- `git checkout -- myfile` will discard all changes with respect to `myfile`

[1] However, going back to an earlier commit leaves you in a "detached HEAD state".

git checkout (cont.)

- `git checkout -- .` (or use `git restore .`) will discard all changes in your working directory, which can include multiple files (remember the dot `.` from my Computer Literacy slides)
- For more information see https://www.atlassian.com/git/tutorials/using-branches/git-checkout

git checkout: Example I

Let's start changing the content of test.R. For instance, remove line 3 (`# Always start with a dumb comment`). First, let's print the original file content again:

```
cd e:/tmp/git_test_folder
cat test.R
```

```
## l1: # Branch: main
## l2: # Author: BW
## l3: # Always start with a dumb comment
## l4: x <- c(1:10)
## l5: mean(x)
## l6: var(x)
## l7: sum(x)
```

```
remove_line("e:/tmp/git_test_folder/test.R", 3)
```

git checkout: Example II

Print out the new code file (remember, the comment line (3. line) has been removed).

```
## l1: # Branch: main
## l2: # Author: BW
## l4: x <- c(1:10)
## l5: mean(x)
## l6: var(x)
## l7: sum(x)
```

Let's check `git status` to see how our repository is doing and what has changed... the important part is `modified: test.R`

```
## On branch 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:   test.R
##
## no changes added to commit (use "git add" and/or "git commit -a")
```

git checkout: Example III

Run `git checkout`...

```
cd e:/tmp/git_test_folder
git checkout -- test.R
```

Voilà, our beloved comment (line 3) has been risen from the dead...

```
## l1: # Branch: main
## l2: # Author: BW
## l3: # Always start with a dumb comment
## l4: x <- c(1:10)
## l5: mean(x)
## l6: var(x)
## l7: sum(x)
```

(Important: after modifying `test.R` we have not commited any changes, i.e., we did not run `git add` and `git commit`)

`git revert`: Another update to `test.R`

Okay, let's again modify test.R. Now, we do this two times. I will use M1 and M2 to denote these two changes (I will also `add` and `commit` these changes).

Again, print new content of `test.R`.

```
## l1: # M1: First line modified
## l2: # Author: BW
## l3: # Always start with a dumb comment
## l4: x <- c(1:10)
## l5: # M2: A new comment
## l6: var(x)
## l7: sum(x)
```

git log

And, let's see the history via `git log`:

```
cd e:/tmp/git_test_folder
git log --oneline
```

```
## bb02cda add new comment (M2)
## 8867449 add new line (M1)
## 520dd2b new branch testing
## 27b2b6c Initial commit
```

Now, we would like to discard any changes introduced by M2 by using `git revert`.

git revert

Put simply: `git revert` can undo a certain commit and adds a new history to the project.

For more information see https://www.atlassian.com/git/tutorials/undoing-changes/git-revert

Example call: `git revert --no-edit HEAD`

- `HEAD`: Revert the very last commit
- `--no-edit`: I do not want to add a commit message

See https://nulab.com/learn/software-development/git-tutorial/git-collaboration/ for the specification of a commit relative to the most recent commit (HEAD)

git revert: Example I

```
cd e:/tmp/git_test_folder
git log
```

```
## commit bb02cda4ffc30ac2dd506239018188a406ab0884
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:27 2023 +0100
##
##     add new comment (M2)
##
## commit 8867449f8003c9ae8741de5eb7c405b410651a0f
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:27 2023 +0100
##
##     add new line (M1)
##
## commit 520dd2b7a73292775a6a2fd83fdf67e6e46ca3ce
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:25 2023 +0100
##
##     new branch testing
##
## commit 27b2b6c09e1937e7dbb957d54ed2ae628f930c99
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:20 2023 +0100
##
##     Initial commit
```

git revert: Example II

Revert and...

```
cd e:/tmp/git_test_folder
git revert --no-edit HEAD
```

```
## [main ee359d9] Revert "add new comment (M2)"
##  Date: Thu Nov 16 07:41:28 2023 +0100
##  1 file changed, 1 insertion(+), 1 deletion(-)
```

...back to M1.

```
cd e:/tmp/git_test_folder
cat test.R
```

```
## l1: # M1: First line modified
## l2: # Author: BW
## l3: # Always start with a dumb comment
## l4: x <- c(1:10)
## l5: mean(x)
## l6: var(x)
## l7: sum(x)
```

git revert: Example III

```
cd e:/tmp/git_test_folder
git log --oneline
```

```
## ee359d9 Revert "add new comment (M2)"
## bb02cda add new comment (M2)
## 8867449 add new line (M1)
## 520dd2b new branch testing
## 27b2b6c Initial commit
```

git reset

Put simply: `git reset` goes back to a certain commit and discards all later commits

Be very careful with `git reset` and do not use it when working with others!

For more information see
https://www.atlassian.com/git/tutorials/undoing-changes/git-reset

Studying Δs

What has changed at the file level? `git show` and `git diff`

In this chapter we will learn about `git show` and `git diff`, which show differences at the file level. However, for those of you who do not feel comfortable using the command line I highly recommend meld (http://meldmerge.org/).

So far, we have only a few commits. `git log` shows all commits, the SHA1 hash and the respective commit message.

```
cd e:/tmp/git_test_folder
git log
```

```
## commit ee359d9c28c2ba782c3c8f5f628afb4fde32e425
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:28 2023 +0100
##
##     Revert "add new comment (M2)"
##
##     This reverts commit bb02cda4ffc30ac2dd506239018188a406ab0884.
##
## commit bb02cda4ffc30ac2dd506239018188a406ab0884
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:27 2023 +0100
##
##     add new comment (M2)
##
## commit 8867449f8003c9ae8741de5eb7c405b410651a0f
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:27 2023 +0100
##
##     add new line (M1)
##
## commit 520dd2b7a73292775a6a2fd83fdf67e6e46ca3ce
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:25 2023 +0100
##
```

A brief intro to the unified diff format

Using `git show` without any additional arguments shows the differences between the last commit and HEAD. The output follows the so called "unified diff format" (UDF). A good introduction of UDF ist provided by https://www.gnu.org/software/diffutils/manual/html_node/Detailed-Unified.html#Detailed-Unified. The following is mostly copy-and-paste from the aforementioned source. It is also imported to note that UDF utilizes so-called (c)hunks to describe changes. A hunk is a paragraph separated by an empty line.

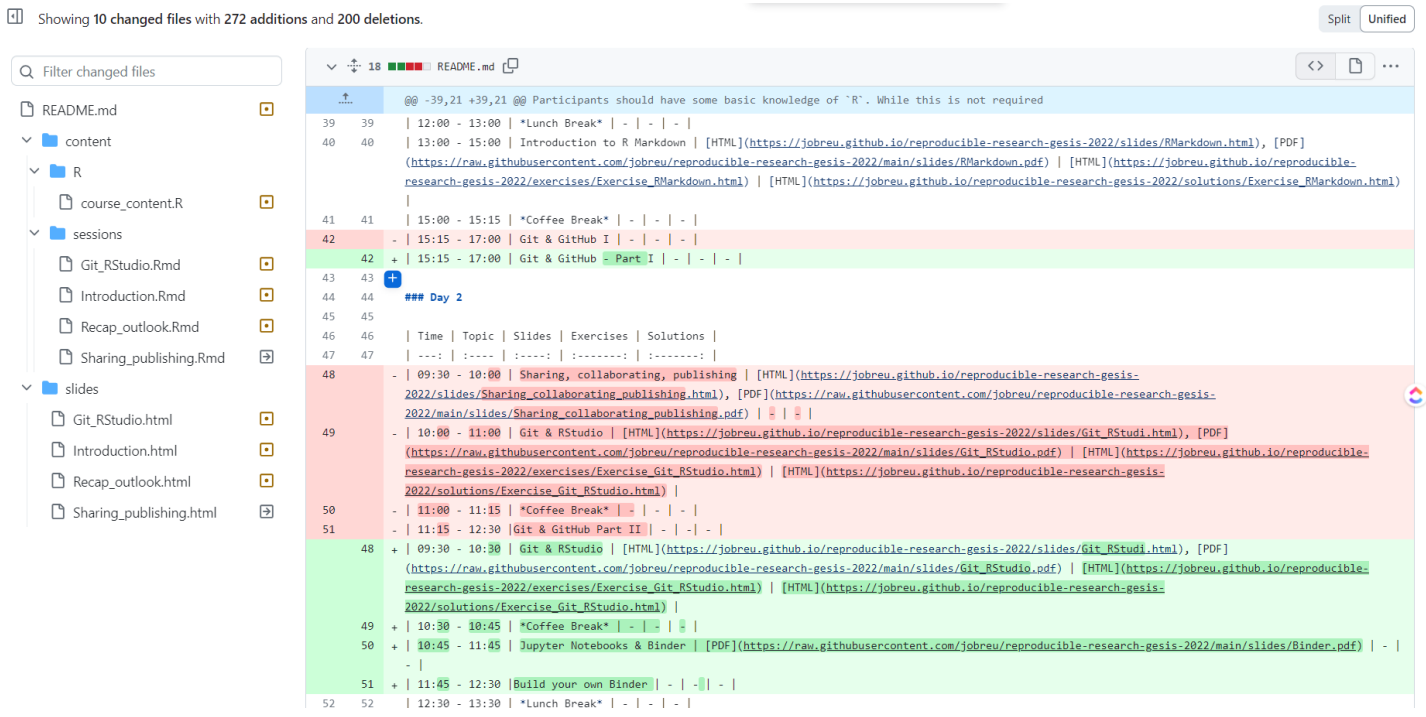
git show

```
cd e:/tmp/git_test_folder
git show
```

```
## commit ee359d9c28c2ba782c3c8f5f628afb4fde32e425
## Author: Bernd Weiss <xx@www.com>
## Date:   Thu Nov 16 07:41:28 2023 +0100
##
##     Revert "add new comment (M2)"
##
##     This reverts commit bb02cda4ffc30ac2dd506239018188a406ab0884.
##
## diff --git a/test.R b/test.R
## index 0084571..c8dd63f 100644
## --- a/test.R
## +++ b/test.R
## @@ -2,6 +2,6 @@ l1: # M1: First line modified
##  l2: # Author: BW
##  l3: # Always start with a dumb comment
##  l4: x <- c(1:10)
## -l5: # M2: A new comment
## +l5: mean(x)
##  l6: var(x)
##  l7: sum(x)
## \ No newline at end of file
```

Who uses `git show` anyway?

Frankly, I go to GitHub or GitLab and check the respective differences between files...



More on GitHub: Inviting collaborators

Adding collaborators to your GitHub repo

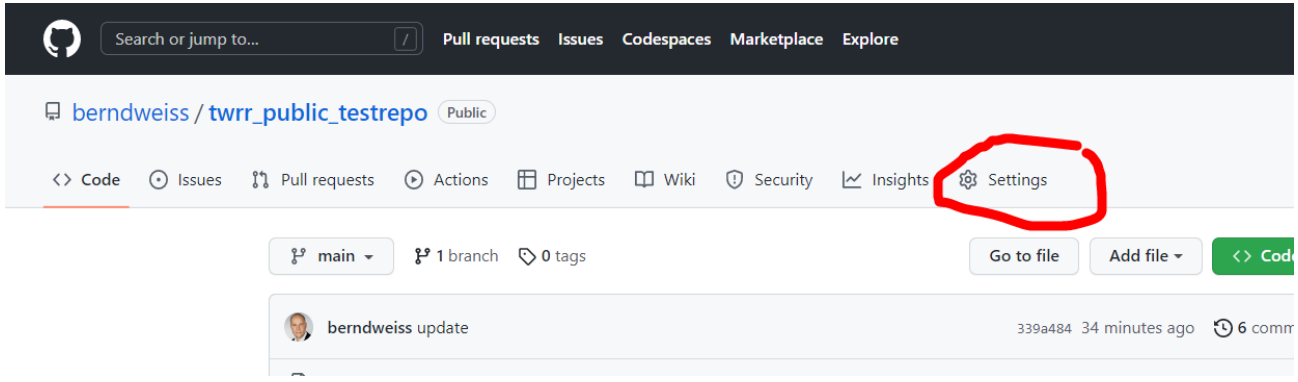
- Adding collaborators works only for GitHub repositories that you own (or have access to and the respective rights)
- GitHub provides a lot of collaboration features
  - Edit files in browser
  - Change highlighting and commenting
  - Interactive revise-and-resubmit workflow
  - Issue tracker (to-do list and discussion)
  - ...

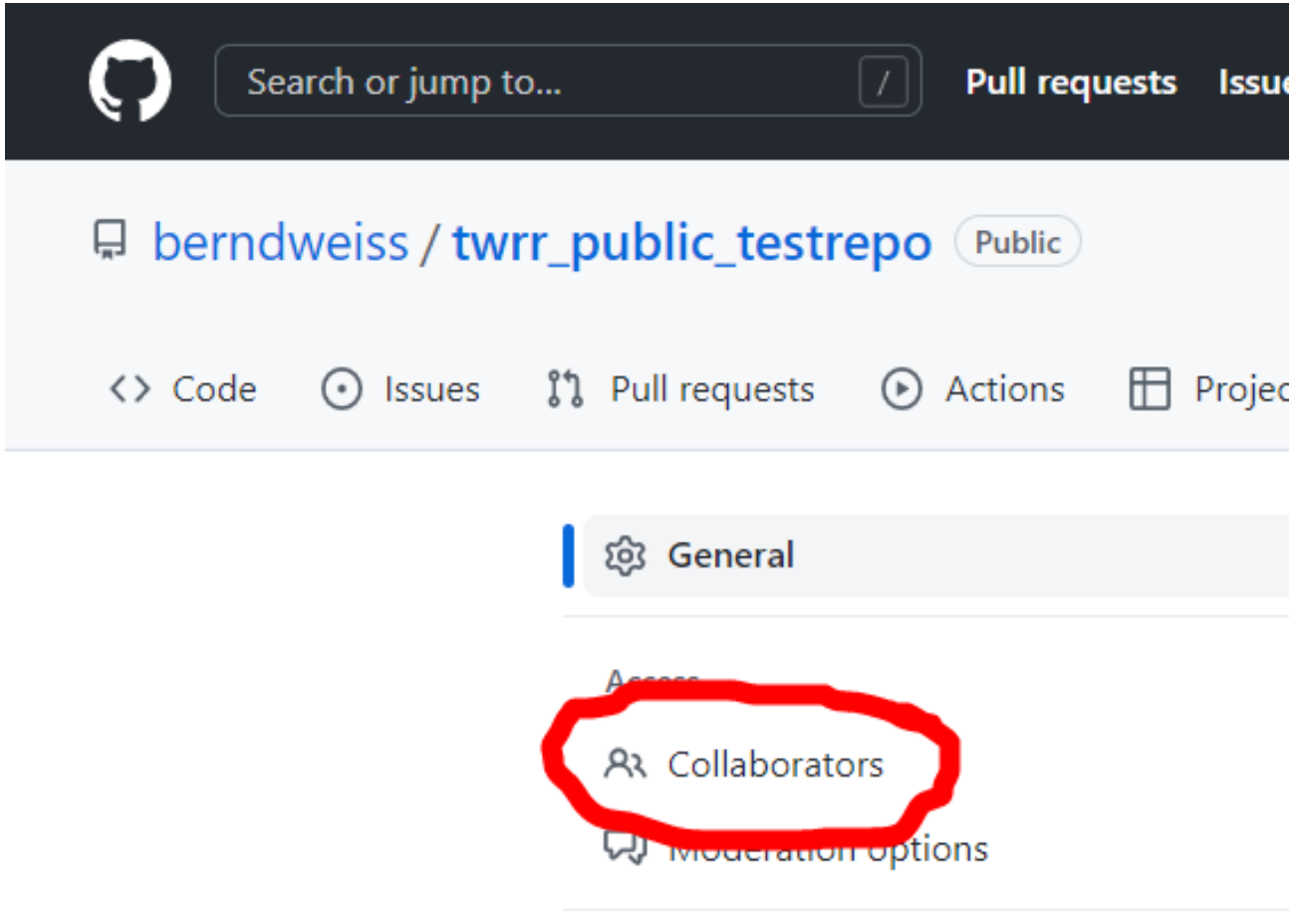
(Source: http://frederikaust.com/reproducible-research-practices-workshop/slides/6_github_collaboration.html#5)

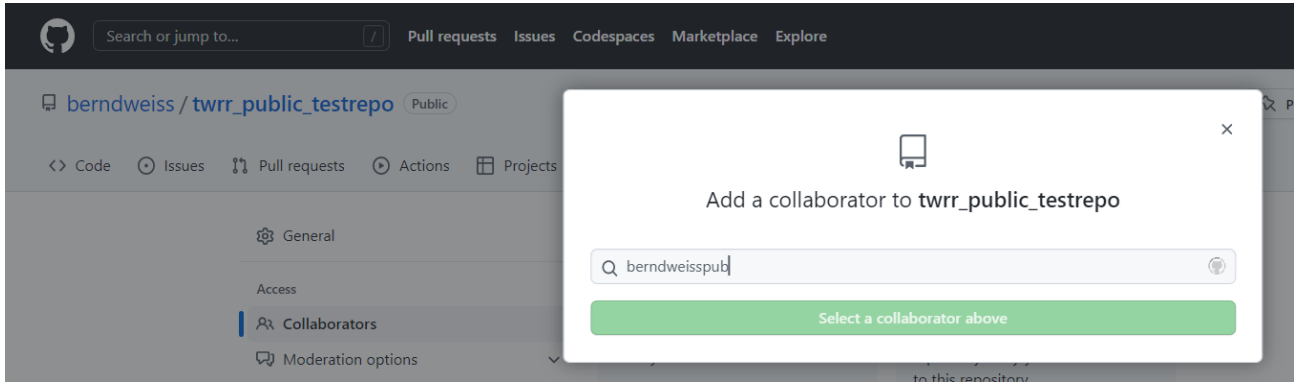
Workflows for collaboration

- Adding changes to a repo without prior review
  - Push directly to main branch on GitHub
  - You need be an invited collaborator
- Suggest changes with review (pull request)
  - Create a new branch ("parallel universe" of repository)
  - You can be an invited collaborator or a complete stranger
- Edits can be made directly on GitHub or locally on your computer

(Source: http://frederikaust.com/reproducible-research-practices-workshop/slides/6_github_collaboration.html#8)


Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
berndweiss / twrr_public_testrepo
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
1 branch
0 tags
Go to file
Add file
berndweiss update
339a484 34 minutes ago
6 comm


Search or jump to...
/
Pull requests
Issue
berndweiss / twrr_public_testrepo
Public
Code
Issues
Pull requests
Actions
Projec
General
Collaborators

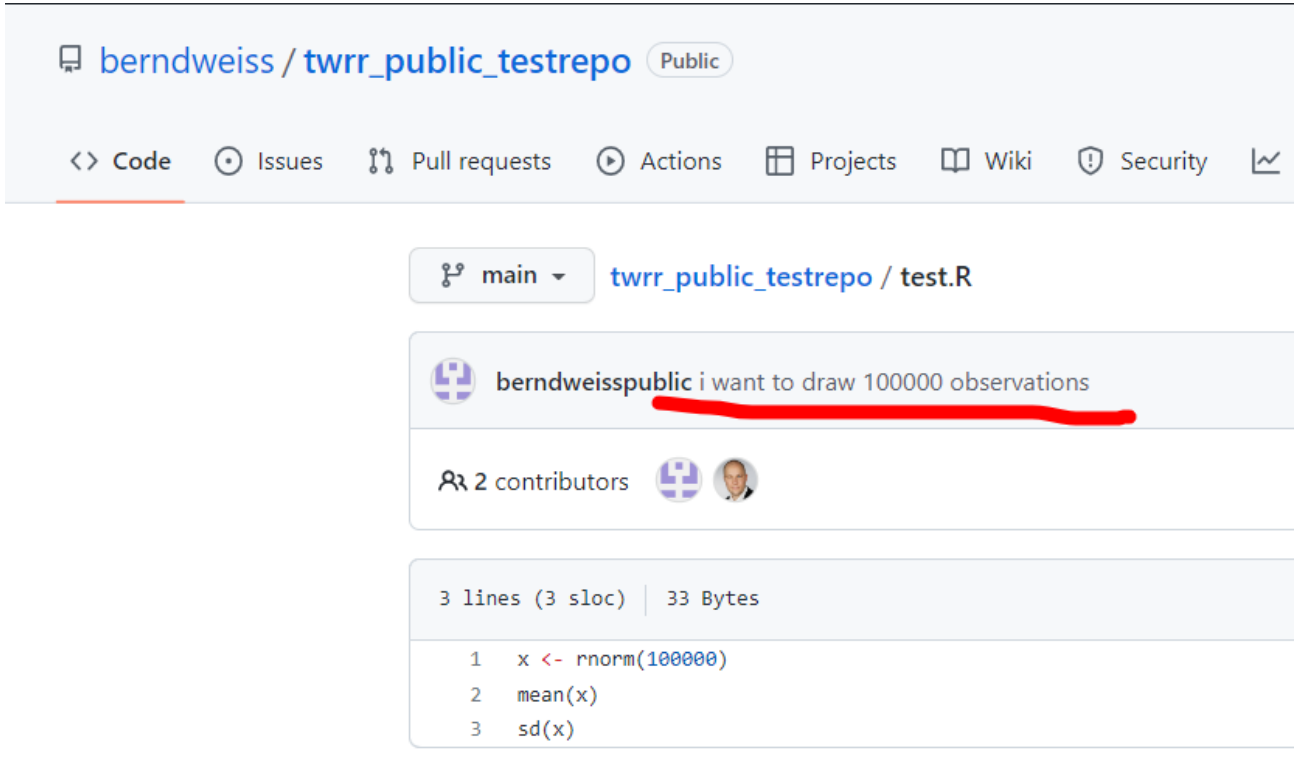

Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
berndweiss / twrr_public_testrepo
Public
Code
Issues
Pull requests
Actions
Projects
General
Access
Collaborators
Moderation options
Add a collaborator to twrr_public_testrepo
berndweisspub
Select a collaborator above

More on GitHub: Merge conflicts

Merge conflicts

- Merge conflicts occur when there are two competing changes that affect the same file *and* the same lines in that same file; or if one person decided to delete it while the other person decided to modify it
- Git will inform you about a merge conflict and will indicate the two competing changes in a file

(source: https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts)


berndweiss / twrr_public_testrepo
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
main
twrr_public_testrepo / test.R
berndweisspublic i want to draw 100000 observations
2 contributors
3 lines (3 sloc) 33 Bytes
1 x <- rnorm(100000)
2 mean(x)
3 sd(x)

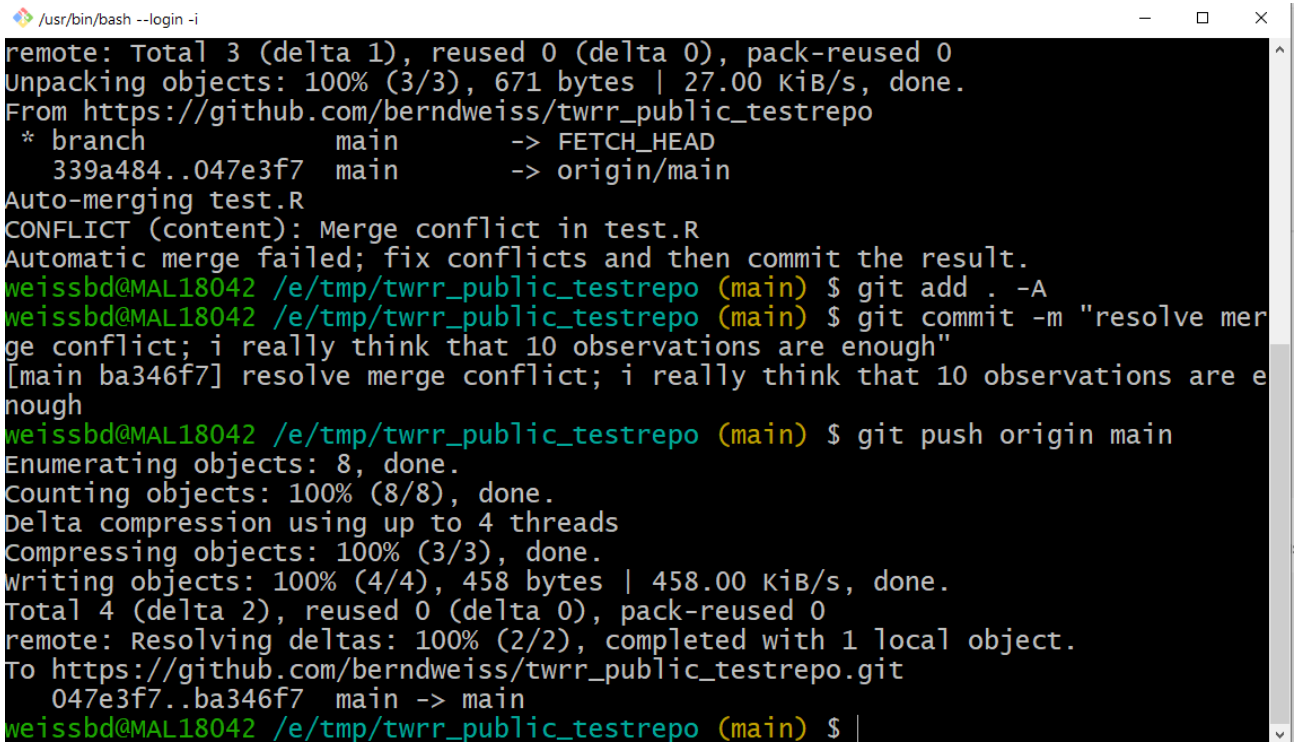
```
nothing to commit, working tree clean
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $ git add . -A
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $ git commit -m "10 observat
ions are enough"
[main fe79328] 10 observations are enough
 1 file changed, 1 insertion(+), 1 deletion(-)
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $ git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 671 bytes | 27.00 KiB/s, done.
From https://github.com/berndweiss/twrr_public_testrepo
 * branch            main       -> FETCH_HEAD
   339a484..047e3f7  main       -> origin/main
Auto-merging test.R
CONFLICT (content): Merge conflict in test.R
Automatic merge failed; fix conflicts and then commit the result.
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $
```

An example of a merge conflict

- This is how a merge conflict looks like in the file `test.R`:

```
<<<<<<< HEAD
x <- rnorm(10)
=======
x <- rnorm(100000)
>>>>>>> 047e3f7a00a5541622e5a40dc342df3af0591838
mean(x)
sd(x)
```

- A merge conflict only affects the developer who is causing a merge conflict
- You have to resolve the merge conflict by editing the respective file(s) (then add and commit the changes) (remove the `<<<<<<<`, `=======`, `>>>>>>>` and save the file)


/usr/bin/bash --login -i
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 671 bytes | 27.00 KiB/s, done.
From https://github.com/berndweiss/twrr_public_testrepo
 * branch   main   -> FETCH_HEAD
   339a484..047e3f7 main   -> origin/main
Auto-merging test.R
CONFLICT (content): Merge conflict in test.R
Automatic merge failed; fix conflicts and then commit the result.
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $ git add . -A
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $ git commit -m "resolve mer
ge conflict; i really think that 10 observations are enough"
[main ba346f7] resolve merge conflict; i really think that 10 observations are e
nough
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $ git push origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 458 bytes | 458.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/berndweiss/twrr_public_testrepo.git
   047e3f7..ba346f7 main -> main
weissbd@MAL18042 /e/tmp/twrr_public_testrepo (main) $

(!) Exercise: Create a merge conflict with yourself

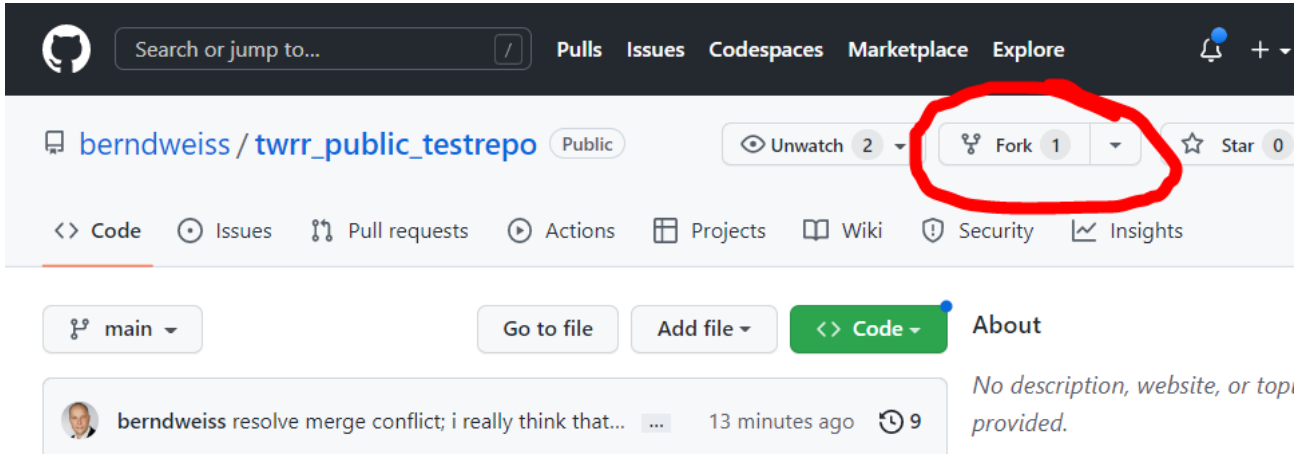
- Create a local Git repo and add a simple text file, e.g., via `echo "123456" > test.txt`
- Create a new repo on GitHub, copy `git remote add ...` and add remote branch
- Commit everything locally, and push it to Github
- Now comes the fun part:
  - Edit the file on GitHub and commit
  - Edit the file in your local Git repo and commit all changes
  - Do a `git pull origin main`
- If everything went, uh, well, then you should see the following error messsage: `Automatic merge failed; fix conflicts and then commit the result.`

More on GitHub: Forking

Forking

- Forking refers to the process of creating a personal copy of someone else's project
- Forking works only for public repositories (or, you have been invited to a private repository)
- In order to contribute to another (public) repository via *pull requests*, you first need to fork the respective repository

(Source: https://docs.github.com/en/get-started/quickstart/contributing-to-projects)

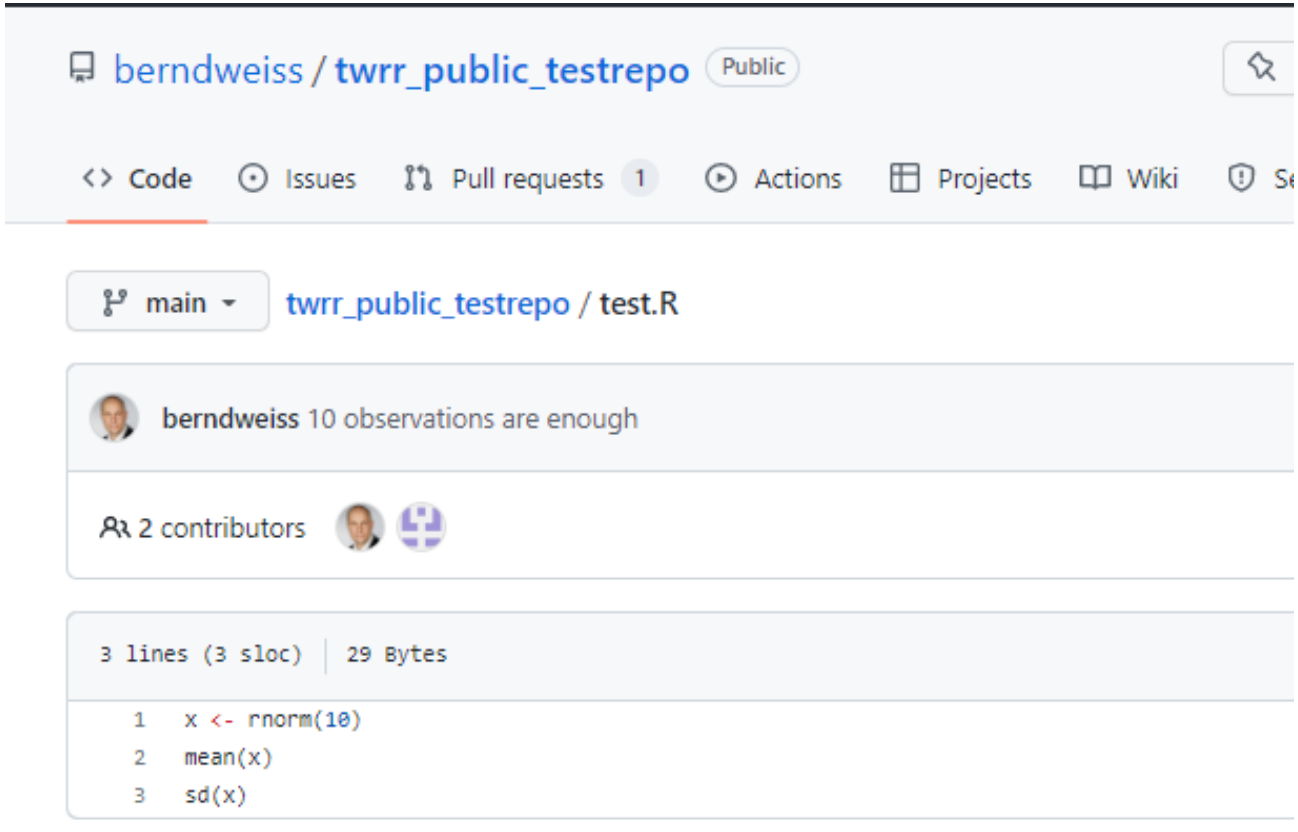

Search or jump to...
Pulls
Issues
Codespaces
Marketplace
Explore
berndweiss / twrr_public_testrepo
Public
Unwatch 2
Fork 1
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
main
Go to file
Add file
Code
About
No description, website, or topi
provided.
berndweiss resolve merge conflict; i really think that...
13 minutes ago
9

More on GitHub: Pull requests

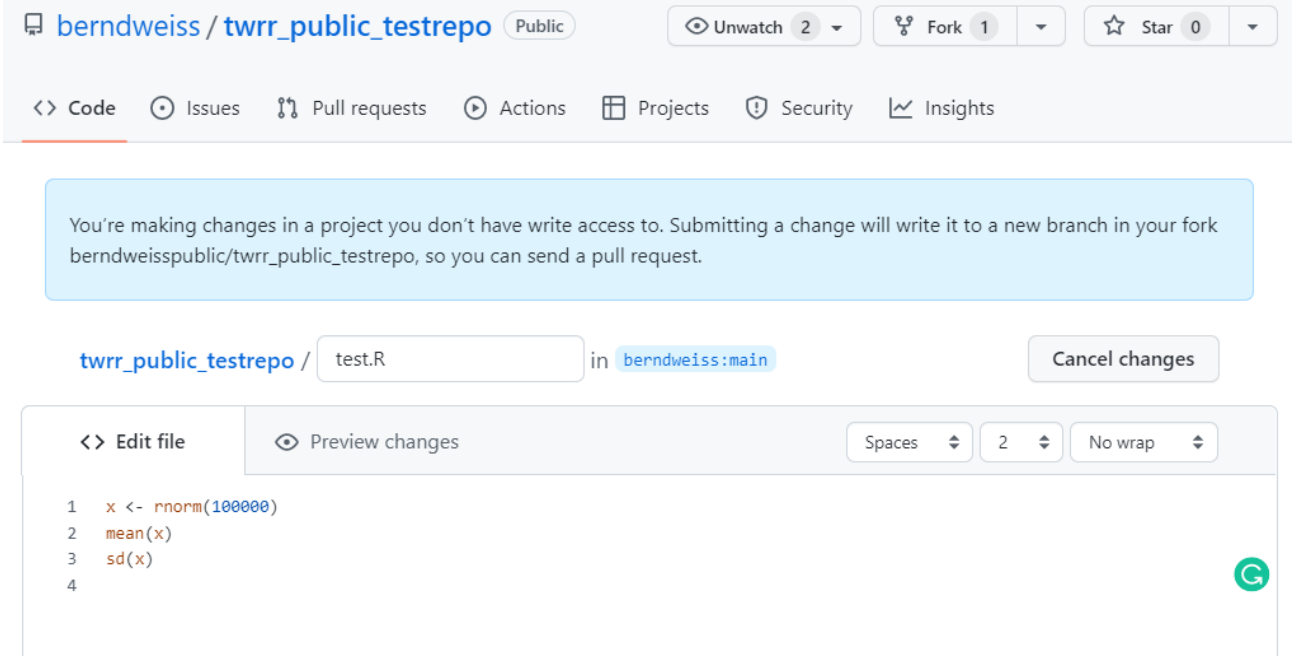
Pull requests on GitHub

- Pull request only work in the context of a web platform such as GitHub or GitLab
- It is a polite / the only way to contribute to another person's GitHub repository
  - If you are a collaborator, then it is a polite way to contribute
  - If you are not a collaborator, then it is the only way to contribute
- Note, the following example is based on
  - two GitHub users: `berndweiss` and `berndweisspublic`
  - both users are not collaborating
  - on each slide I will indicate which GitHub user is currently involved

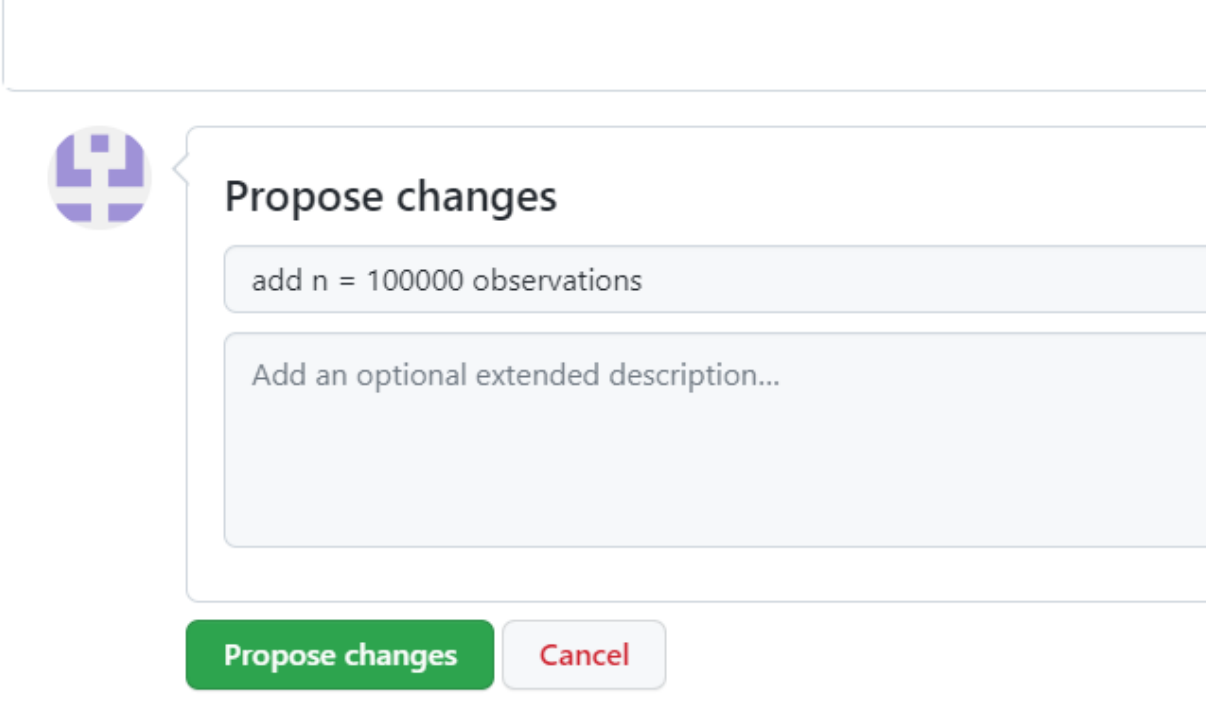
- The current status of the repo (and the file `test.R`) from the perspective of GitHub user `berndweiss`



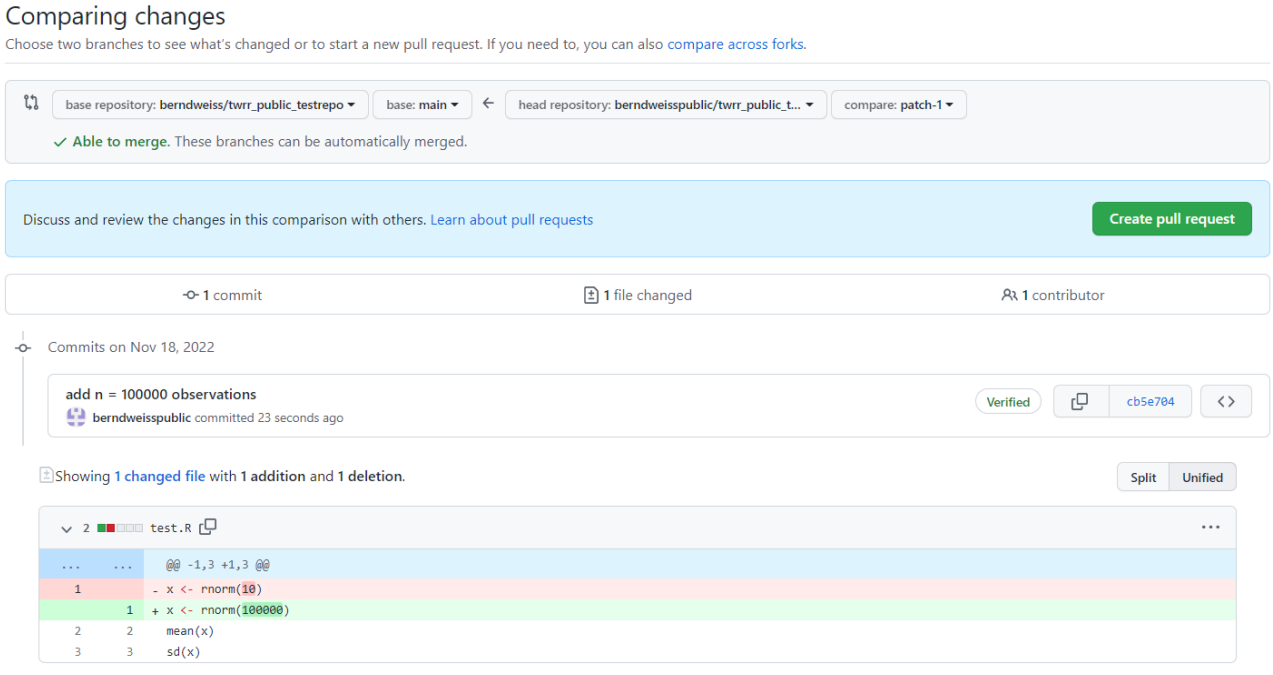
- GitHub user `berndweisspublic` insists of 100000 observations and modified the file accordingly



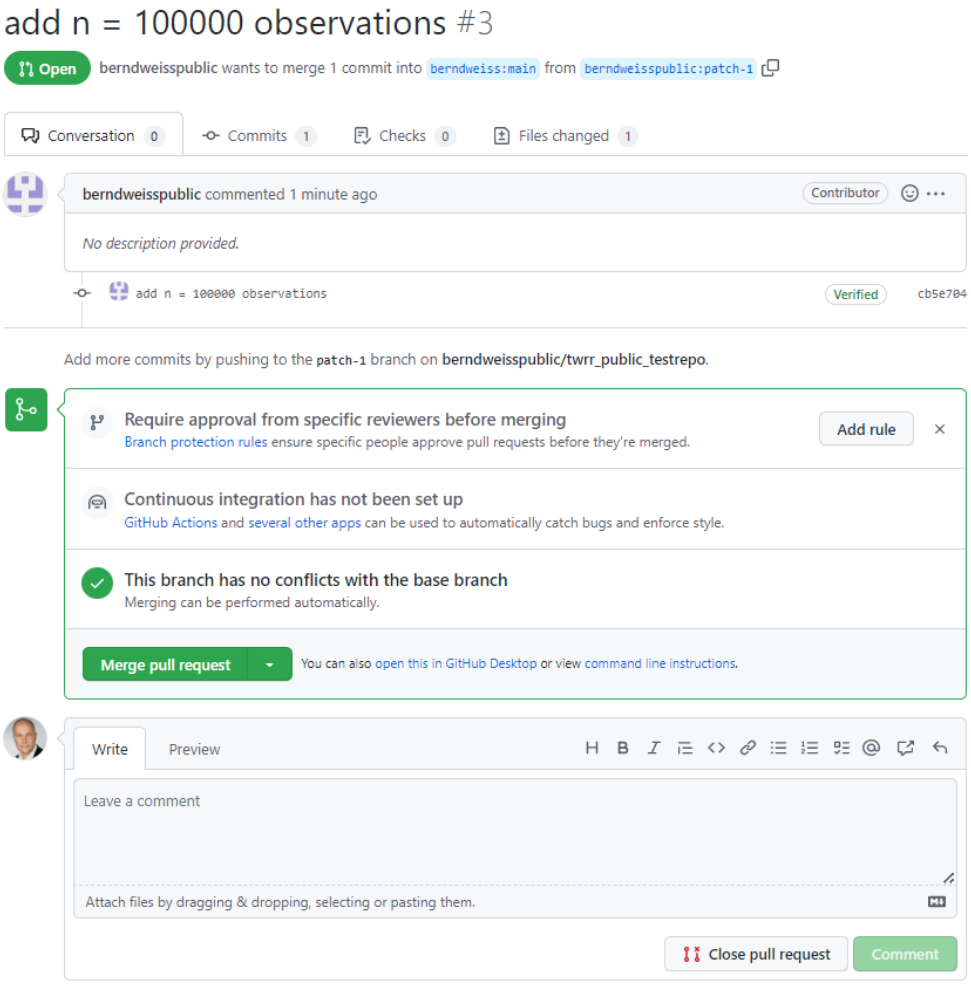
Since GitHub user `berndweisspublic` does not own the repository, he cannot commit the changes but *proposes* changes



The next step is that `berndweisspublic` creates a *pull request*, i.e., asking user `berndweiss` to accept his changes



GitHub user `berndweiss` is informed about a pull request; he can accept the pull request (`merge`) or close the pull request, i.e., deny it



References

Healy, K. (2019, Oktober 4). The Plain Person's Guide to Plain Text Social Science. The Plain Person's Guide to Plain Text Social Science. https://plain-text.co/

Narębski, J. (2016). Mastering Git: Attain expert-level proficiency with Git for enhanced productivity and efficient collaboration by mastering advanced distributed version control features. Packt Publishing.