

# Git wise!

An introduction version control with Git:  
Branching and teamwork

19 May 2026  
Linus Band

Version 0.9.1

# Course overview

1. Short introduction + tiny recap
2. Branches
3. Remote repositories



# Introductions

1. Who am I?
  - a) Linus Band ([linus.band@ru.nl](mailto:linus.band@ru.nl))
  - b) Research software trainer and data steward at the RU's Digital Competence Centre.
2. Who are you?

# What, When, Who, Why???

## Version control

- **What:** It keeps track of the changes made to a file when you edit it.
- **When:** It records the date and time when each change was made.
- **Who:** It keeps a record of who made each change.
- **Why:** It allows you to add a note explaining why you made a particular change.

# Introducing...



- **Official Git documentation:** <https://git-scm.com/>
- <https://gitforwindows.org/>
- **Useful course:** <https://lennartwittkuhn.com/version-control-book/>
  - **Basis for this presentation**
    - *All information here is taken from there, unless specified otherwise*
  - **Also has an online repo**

# Before we get started...

- **Keep this cheatsheet handy:** <https://lennartwittkuhn.com/version-control-book/misc/cheatsheet.html>

# What are branches?

- If you followed session 1 of this course, you've used only the **main** branch
  - Branching creates a copy of a project:
    - Each branch keeps own history of changes
    - You can have as many branches as you want
    - You can branch off any branch you like
- Allows making commits on that branch only
- You can merge it with another branch, when satisfied

# Why use branches? – 1

- Creating a new feature without breaking existing functionality
  - Allows you to go back to original version to:
    - Compare output
    - Do quick bugfixes
    - Etc.
- When collaborating, avoid getting in each other's way

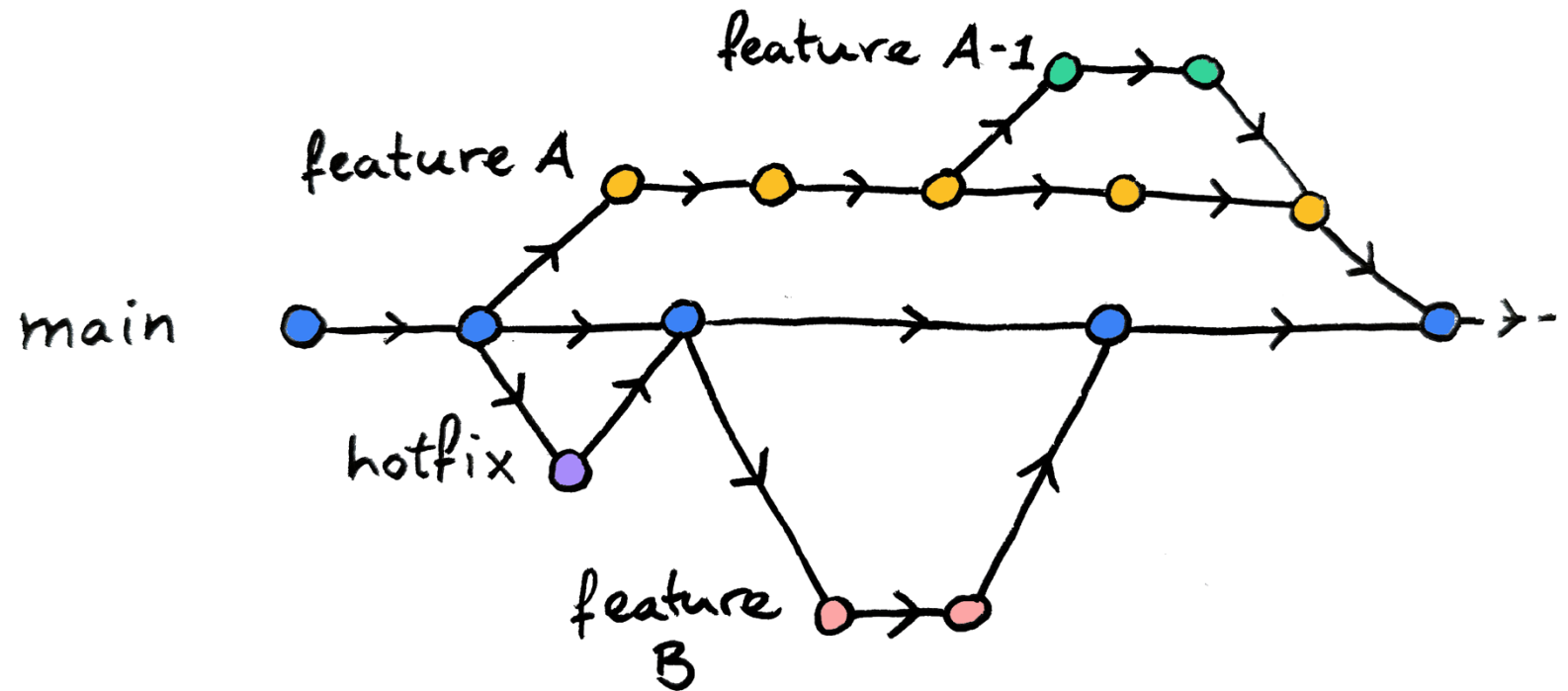




# Why use branches – 2

## Branches can be used:

- To add new features
- To do quick bugfixes/hotfixes
- To simultaneously explore different parameters
- As sandboxes to play around, i.e. test things
- Etc.



Based on the image from [Chapter "Git Branches"](#) of the ["The Turing Way handbook to reproducible, ethical and collaborative data science"](#), used under a [Creative Commons Attribution 4.0 License](#).

# Why use branches – 3

*“Git makes it easy to switch between branches and handle conflicts when merging.*

*With branches, you can work independently and keep your code organized.”*

Source: <https://lennartwittkuhn.com/version-control-book/chapters/branches.html#what-are-branches>

# Basic branch commands

- **git branch --list**: lists all of your branches. \* Indicates your current branch
  - **git branch -l** for short
  - **git branch** if you feel lazy/cool 😎
- **git branch BRANCH-NAME**: creates a new branch with the name BRANCH-NAME
- **git switch BRANCH-NAME**: switches to branch named BRANCH-NAME
  - **git switch -c BRANCH-NAME**: creates a new branch with name BRANCH-NAME **AND** switches to it
  - **git checkout BRANCH-NAME**: does the same (**switch** is less ambiguous, **checkout** also has other uses)
    - **git checkout -b BRANCH-NAME**: same as git switch -c BRANCHNAME

**Note**: you cannot switch branches before making at least one commit on your current branch!

# Exercise 1 – creating branches

1. Create a new folder (in you **tmp** directory for example)
2. Create a git repo there
3. Create a README.md and commit it
4. Create new branch
5. Switch to your new branch
6. List your branches and marvel at the possibilities


# Merging branches – 1

- When work on your branch is ready: **time to merge!**
- Go to the target branch you want to merge into (often **main**)
- **git merge BRANCH-NAME**




# Merging branches – 2

- When work on your branch is ready: **time to merge!**
- Go to the target branch you want to merge into (often **main**)
- **git merge BRANCH-NAME**
  - If there are no changes
  - If there are changes



```
test4_fast-forward git:(main) git merge new-feature
Already up to date.
test4_fast-forward git:(main)
```



```
test4_fast-forward git:(main) git merge new-feature
Updating bb4f418..0a56b99
Fast-forward
 file3 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file3
test4_fast-forward git:(main)
```

# Exercise 2 – merging branches

1. You should be in the new branch you created for exercise 1
2. Create a new file with a bit of text in it
3. Commit that file
4. Switch back to main
5. Merge your new branch into main

# Merge conflicts ✖✖

- When both branches have conflicting changes in the same file
- Git won't know which version to choose
- It needs **your help!**
- How to avoid? No... **minimise!** (*You are bound to run into them...*)
  - Have **a clear workflow!** (*See section on remote repositories*)

```
[→ test4_fast-forward git:(main) git merge new-feature
Auto-merging file1
CONFLICT (content): Merge conflict in file1
Automatic merge failed; fix conflicts and then commit the result.
→ test4_fast-forward git:(main) x
```



# Exercise 3

1. In your main branch open your text file from exercise 2
2. Add a line at the bottom saying "How long til the break?"
3. Commit this change
4. Switch to your existing other branch (this should still have the original text only)
5. Add a line at the bottom saying "My brain hurts, I need coffee..."
6. Commit this change
7. Switch to main
8. Merge the 'new' branch into main
9. Huzzah! Merge conflict!

# Fixing merge conflicts – 1

- How to fix?
  - Git does help you!
  - Fix conflicts and **git commit**!

**<<<<<< HEAD**      your current branch's  
version

**=====**            marks where the version  
from the other branch  
starts

**>>>>>> new-feature** is the end of the version  
from the other branch  
(**new-feature** will be the  
name of your specific  
branch)

```
test4_fast-forward git:(main) ✕ git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   file1

no changes added to commit (use "git add" and/or "git commit -a")
test4_fast-forward git:(main) ✕
```

```
test4_fast-forward git:(main) ✕ cat file1
text for new file
<<<<<<< HEAD
How long til the break?
=====
I need coffee!
>>>>>>> new-feature
test4_fast-forward git:(main) ✕
```

# Fixing merge conflicts – 2

1. Edit the conflicted file
2. Add it to the staging area
3. Commit it to resolve the conflict
4. Rejoice!

```
test4_fast-forward git:(main) x cat file1
text for new file
<<<<<< HEAD
How long til the break?
=====
I need coffee!
>>>>>> new-feature
test4_fast-forward git:(main) x
```

```
1 text for new file
2 How long til the break? I need coffee!
3
```

# Exercise 4

1. Go to the merge conflict from exercise 3
2. Resolve it 😎

# Cleaning up your branches

- Keep your branches clean
  - Delete branches you are done with
  - **git branch -d <branch-name>**
  - Check if it was deleted with **git branch -l** (or **git branch --list** in full)

# Switching with uncommitted changes

- You can switch branches with uncommitted changes **if they do not conflict** with target branch
  - Useful if you started working in the wrong branch
- If they do conflict, Git will prevent you from switching

## Solutions

- Commit your changes before switching
- Stash your changes (see this chapter in the Version Control Book)
- Discard your changes
  - Use **git reset** for this

```
error: Your local changes to the following files would be overwritten by
checkout:
    example.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

<https://lennartwittkuhn.com/version-control-book/chapters/branches.html#switching-branches-with-uncommitted-changes>

# Remote repos – what?

## Is a:

- Code hosting service
- Collaborative software development environment



## Basically:

- Hosts your code
- Allows you to share it (or not)
- Allows to work on it with others



# Remote repos – where?

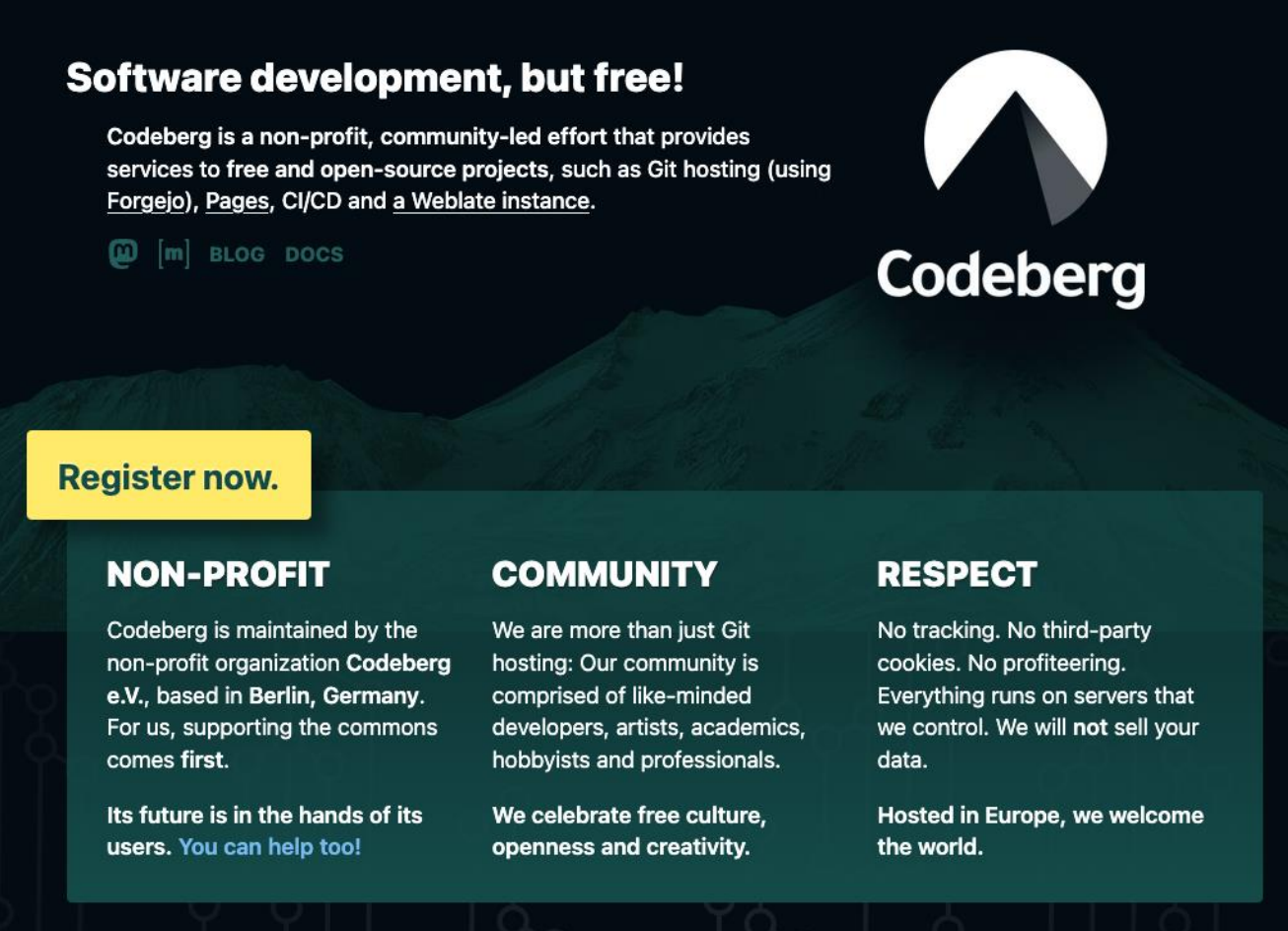
- Non-profit (in EU)
- Privately owned (in US)
- Open source  
**Note:** only self-hosted GitLab is **OS**!
- At the moment (2026-05-18) the privately owned solutions are more feature rich.
- For most applications, **Codeberg** should be sufficient!





# Registering and connecting

1. Go to [codeberg.org/](https://codeberg.org/)
2. Create an account

A screenshot of the Codeberg website. The background is dark with a mountain range silhouette. At the top right is the Codeberg logo, a white circle with a dark triangle inside. Below the logo is the word "Codeberg" in white. To the left of the logo, the text "Software development, but free!" is displayed in white. Below this, a paragraph describes Codeberg as a non-profit, community-led effort providing services like Git hosting, Pages, CI/CD, and a Weblate instance. Below the paragraph are links for GitHub, Matrix, Blog, and Docs. A yellow button with the text "Register now." is positioned on the left side. At the bottom, there are three columns of text: "NON-PROFIT", "COMMUNITY", and "RESPECT", each with a description of Codeberg's values and mission.

**Software development, but free!**

Codeberg is a non-profit, community-led effort that provides services to free and open-source projects, such as Git hosting (using Forgejo), Pages, CI/CD and a Weblate instance.

[\[u\]](#) [\[m\]](#) [BLOG](#) [DOCS](#)

**Codeberg**

**Register now.**

<b>NON-PROFIT</b>	<b>COMMUNITY</b>	<b>RESPECT</b>
Codeberg is maintained by the non-profit organization <b>Codeberg e.V.</b> , based in <b>Berlin, Germany</b> . For us, supporting the commons comes <b>first</b> .	We are more than just Git hosting: Our community is comprised of like-minded developers, artists, academics, hobbyists and professionals.	No tracking. No third-party cookies. No profiteering. Everything runs on servers that we control. We will <b>not</b> sell your data.
Its future is in the hands of its users. <a href="#">You can help too!</a>	We celebrate free culture, openness and creativity.	Hosted in Europe, we welcome the world.

# SSH – what?

## SSH (Secure Shell)

*A secure, **encrypted** way for **authenticating** and **communicating** with a **remote server***

A **server** may be:

- A remote **Git** repository;
- A **High Performance Cluster** (HPC);
- Or any other server.

SSH uses a **key pair**:

- a **private key** (kept secret on your machine)
- a **public key** (shared with the server)

# SSH – why?

- **Security:** strong and encrypted authentication
- **Convenient:** push and pull without entering password each time
- **Access control:** revoke keys without changing passwords or accounts (e.g. in case of device loss)
- **Scriptability:** SSH keys can be integrated in CI/CD
- **Integrity & authenticity:** make sure you are connecting to intended server and there is no tampering
- **Support:** most Git hosts and servers support SSH and it works across firewalls.

# SSH – how?

1. Open <https://docs.codeberg.org/security/ssh-key/>
  - Or, go to <https://docs.codeberg.org>
    - In the left menu bar, scroll to the heading **Security** > **Adding an SSH key to your account**
2. Follow the steps under **Generating an SSH key pair**
3. Follow the steps under the **Add the SSH key to Codeberg**


# SSH – custom name

## NOTE

- If you already have an **ed25519 key pair** and you want a **custom named** one for Codeberg, you can follow the steps under the documentation link in the previous slide.
- **HOWEVER!** If you try to add that **custom named key pair** to your Codeberg account, you may get an **ERROR** that you don't have access rights

```
[➔ git-wise-cursus git clone ssh://git@codeberg.org/gwron-epig/test-repo-codeber]
g-ssh-id.git
Cloning into 'test-repo-codeberg-ssh-id'...
Connection closed by 217.197.84.140 port 22
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```



## Why

- **git clone** uses SSH in the background, which by default will try a standard set of keys.
- If your custom named key pair is not among these, you won't get access!

## Solution

1. Under your .ssh folder, open (or create) your **config** file
2. Add these lines, but make sure the **IdentityFile** matches yours (path and name)
3. Save, close, restart terminal and try cloning again.

```
Host codeberg.org
HostName codeberg.org
User git
IdentityFile ~/.ssh/id_ed25519_codeberg
```

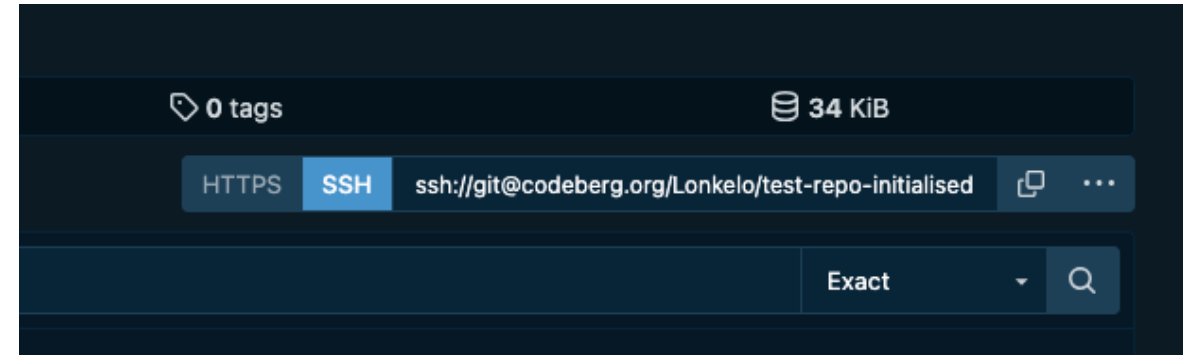
# Creating a repo

1. Open <https://docs.codeberg.org/getting-started/first-repository/>
  - Or, go to <https://docs.codeberg.org>
    - In the left menu bar, scroll to the heading **Getting Started with Codeberg > Your First Repository**
2. Follow the steps :P

## Note:

- You can create a (Git) initialised repo, which you can then clone to your device;
- Or a blank repository that you can push your locally initialised repo to.

# Accessing a repo



- Only **collaborators** have access to a repo and **can directly commit changes**
- Any Codeberg user can contribute through **issues** and **pull requests** (if not **private**)
- To add a **collaborator**:
  - In the repo, top-right: **Settings** > **Collaborators**
  - Click **Add Collaborator**
- You can clone a repo you have access to: **git clone SSH-URL** (see image)

```
➜ tmp git clone ssh://git@codeberg.org/Lonkelo/test-repo-initialised.git
Cloning into 'test-repo-initialised'...
The authenticity of host 'codeberg.org (217.197.84.140)' can't be established.
ED25519 key fingerprint is: SHA256:mIlxA9k46MmM6qdJOdMnAQpzGxF4WIVVL+fj+wZbw0g
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'codeberg.org' (ED25519) to the list of known hosts.
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), 5.76 KiB | 2.88 MiB/s, done.
```

# Exercise 5 – Cloning a repo

1. The trainer should have forked this remote repo once for each group.
2. Go to the remote repository for your group:
  1. *Insert link for group 1*
  2. *Insert link for group 2*
  3. *Etc.*
3. Clone it locally
4. Change directory into the cloned repo and run **git remote -v** to check the URL

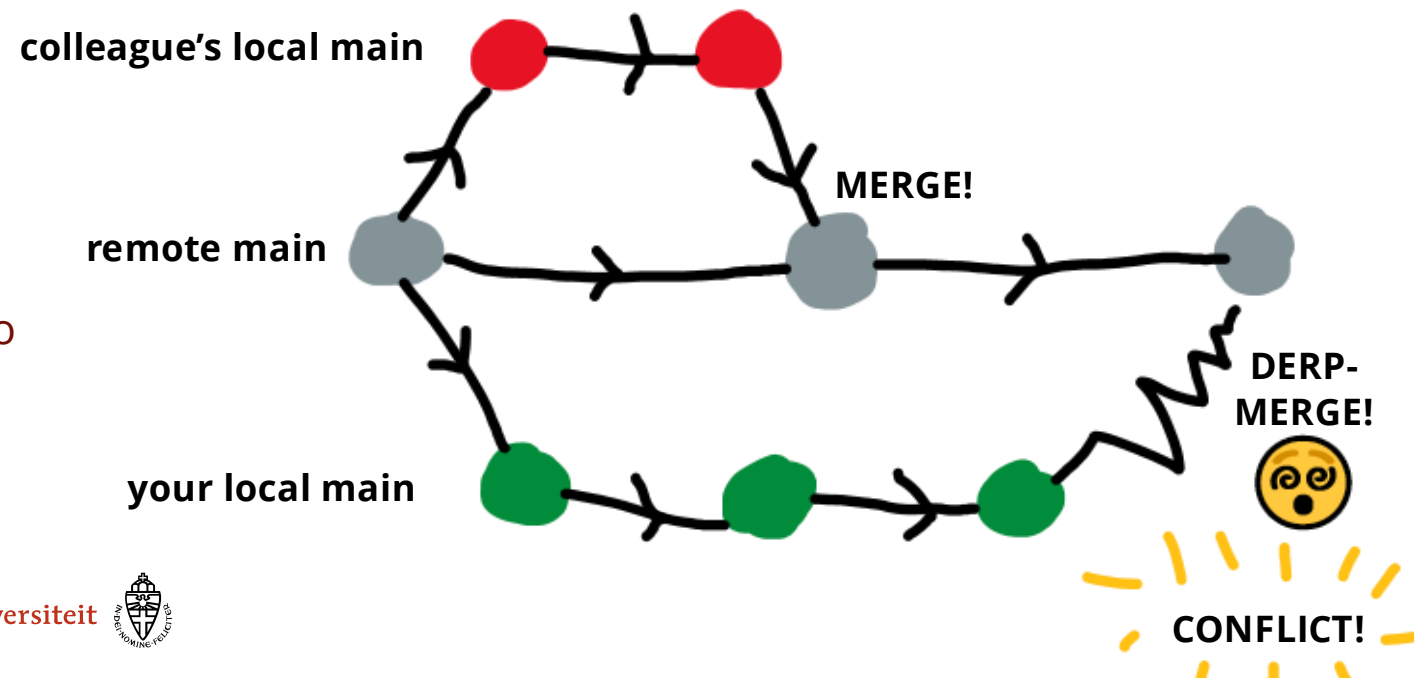


# Pushing and pulling changes – 1

- **git pull** : pulls the latest changes from remote repo and merges them into local branch
- **git push origin <branchname>** : uploads local commits explicitly to remote repo branch of name <branchname>
- **git push** : is shorter and uses your repo's defaults. Note that if this fails, git will normally inform you of what you should do.

## WATCH OUT!

- Your local branch may differ from the remote branch!
- Before pushing or pulling, make sure you have no changes that could clash with the remote



# Pushing and pulling changes – 2

**Solutions** to merge conflicts to and from remote:

- The **hard** way:
  1. Roll back your local changes (**git reset, git stash, etc.**)
  2. Bring local branch up-to-date with remote (**git pull**)
  3. Apply original changes
  4. Try again
- The **better** way: **avoid them** with a proper workflow!

# Pushing and pulling changes – 3

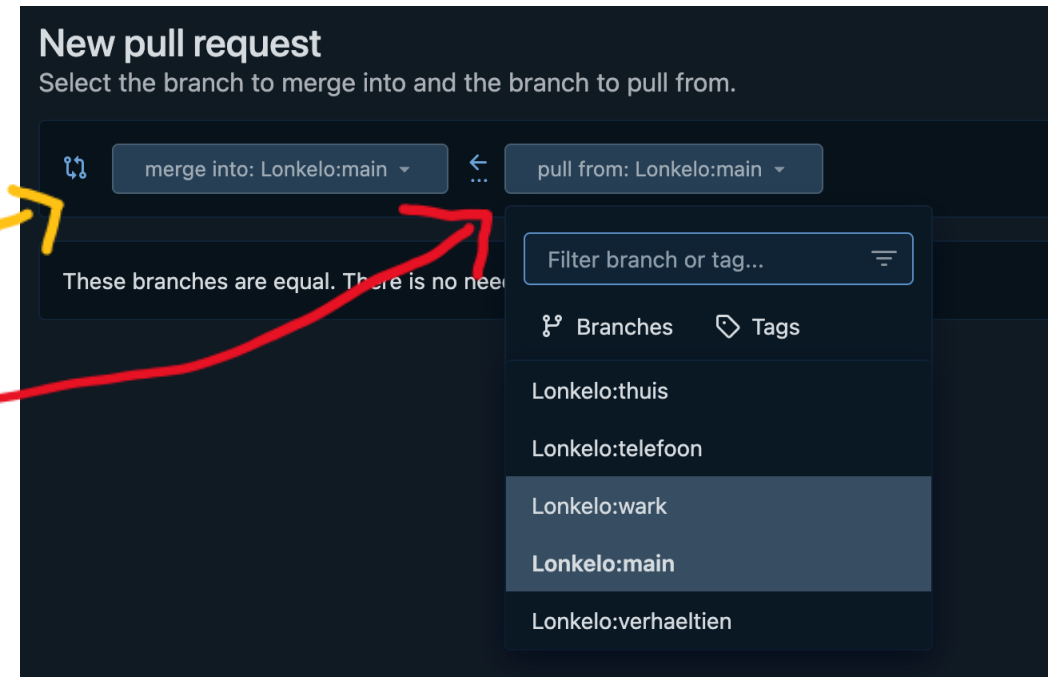
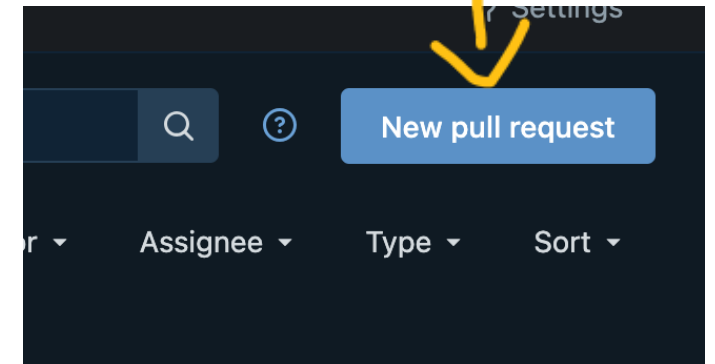
## Proper workflow:

1. Create a branch for your changes, e.g. **git switch -c new-branch-name**
2. Switch to **main** branch, where you branched off
3. Pull changes from **remote**
4. Checkout **new-branch-name**
5. Merge main into **new-branch-name**
6. Push (**git push -u origin new-branch-name**)
7. Got to the online repository interface and **create a pull request!**

# Creating a pull request – 1

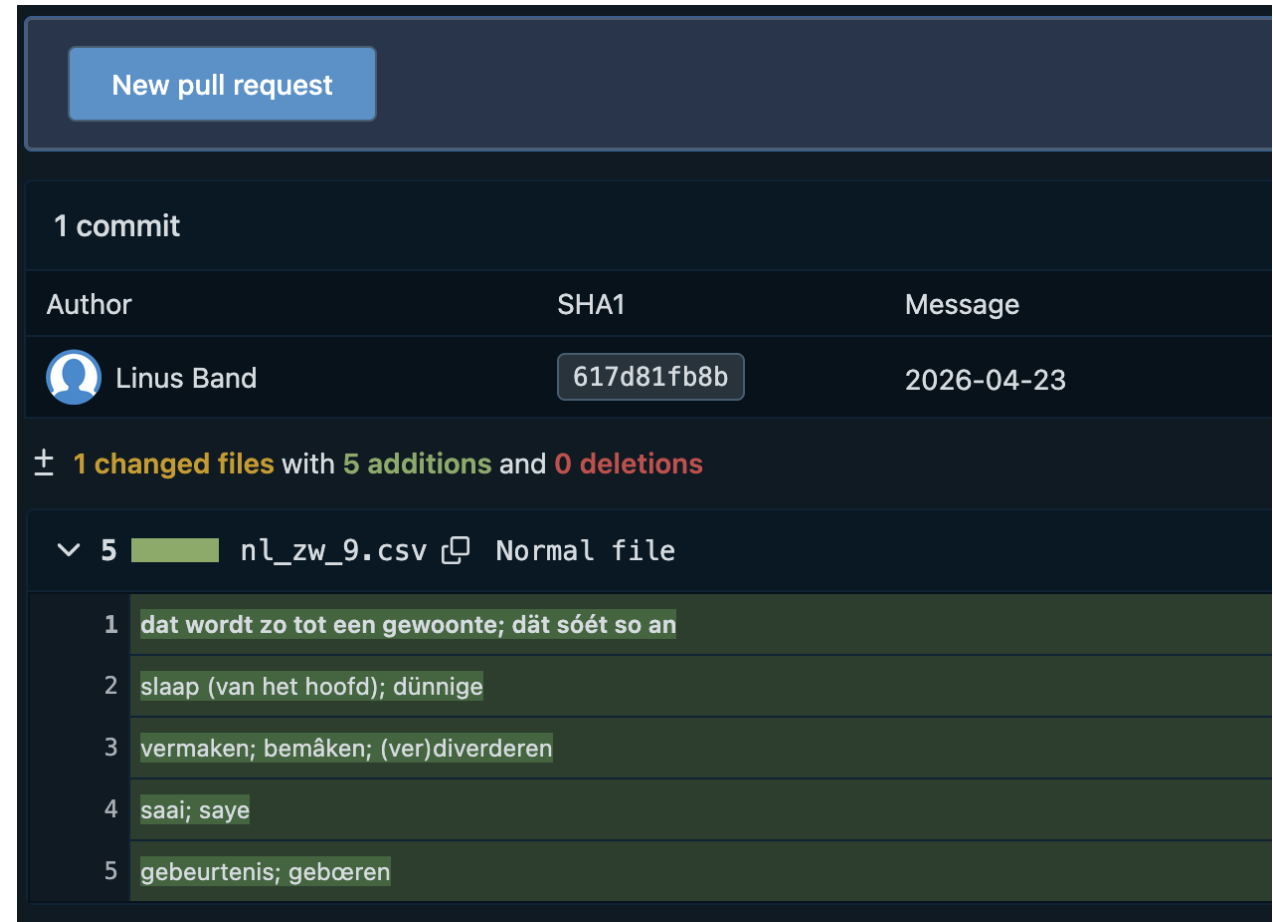


1. Go to **Pull requests** tab
2. Top-right on repo page on Codeberg, click **New pull request**
3. Select the target branch (left) and new branch (right)



# Creating a pull request – 2

1. Check your changes:
  1. Additions in **green**
  2. Removals in **red**
2. Good? Click **New pull request!**
3. In next screen:
  1. Add optional comments for others
  2. Assign **reviewers** (and/or labels and milestones)

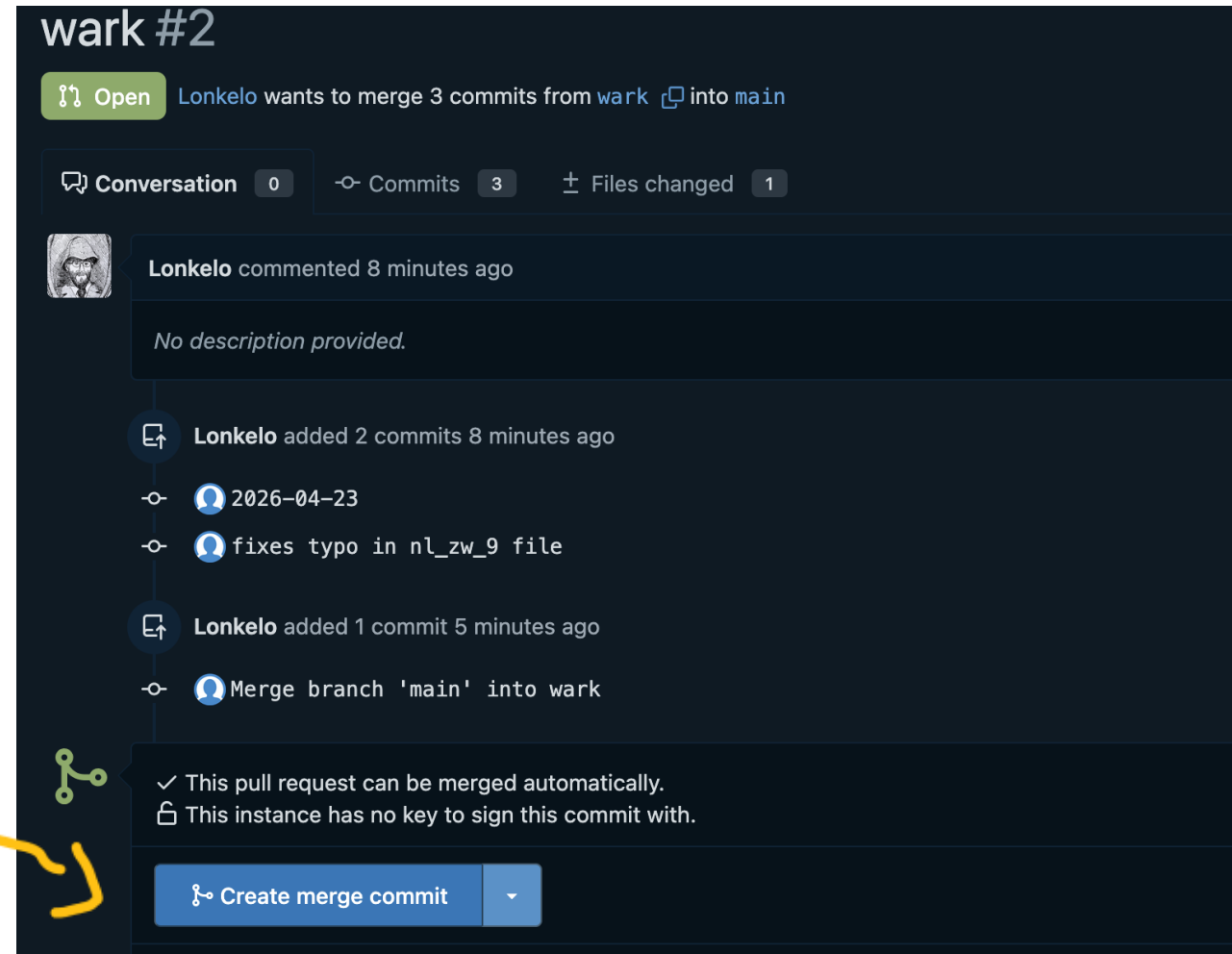


The screenshot shows the GitHub interface for creating a new pull request. At the top, there is a blue button labeled "New pull request". Below this, it indicates "1 commit" and shows the commit details: Author "Linus Band", SHA1 "617d81fb8b", and Message "2026-04-23". A summary shows "1 changed files with 5 additions and 0 deletions". The file "nl\_zw\_9.csv" is listed as a "Normal file". The diff view shows five lines of code with green highlights indicating additions:

Line	Content
1	dat wordt zo tot een gewoonte; dät sóét so an
2	slaap (van het hoofd); dünnige
3	vermaken; bemâken; (ver)diverderen
4	saai; saye
5	gebeurtenis; geboeren

# Creating a pull request – 3

1. The assigned **reviewers** can check code:
  - a. The proposed changes make sense
  - b. The proposed changes work
2. They **approve** the pull request (or not!)
3. Merge pull request by clicking **Create merge commit**
4. Optionally add message
5. Click **Create merge commit** again!
6. **Your target branch has been updated remotely!**
7. Make sure you **pull the merge locally**
8. Clean up any branches you are done with
9. Continue work



# Exercise 6 – creating a merge request 1

*The steps below indicate steps for different group members. Do try to follow these steps together: one person does the step, the other(s) watch and comment.*

1. Person 1 in your group **creates a new branch**
2. Person 1 makes an addition to the *Radboud.md* file (add some trivia or any kind of nonsense you wish ☺ )
3. Person 1 **pushes the changes** to remote and **creates a pull request**
  - Add the other(s) in your group as **reviewer(s)**
4. Person 2 (and 3 if present) **check the pull request** and **approve** it if they agree
5. Decide who gets to merge it: **merge it!**

# Exercise 7 – creating a merge request 2

1. Everyone in your group **creates a new branch**
2. Everyone makes an addition to the *Radboud.md* file (more trivia! More nonsense!)
3. One person **pushes the changes** to remote and **creates a pull request**
  - Add the other(s) in your group as **reviewer(s)**
4. Person 2 (and 3 if present) **check the pull request** and **approve** it if they agree
5. Decide who gets to merge it: **merge it!**
6. The other(s) are now free to **push their changes** and **create a pull request** (*careful now!*)
7. Possible outcomes:
  1. Merge conflict(s)? Excellent practice opportunity!
  2. No hiccups. A bit boring, but well done ;P



*Cool Owl approves.*

**You got...**

*Give yourself  
a pat on the shoulder!*

