Introduction to git - Part B

Contents

1 Goals 3
1.1 Tools 3
2 Basic Operations 5
2.1 Creating a git repository 5
2.2 Clone 6
2.3 Get status 7
2.4 Stage & commit 8
2.5 Add files 10
2.6 Markdown 11
2.7 Push online 12
2.8 Back in time (not the future) 13
2.9 General questions 14
2.10 Tag 14
3 Branch and Merge 15
3.1 First Branch 15
3.2 Second Branch 16
3.3 Merge dev02 17
3.4 Merge dev01 17
3.5 Final Result 18
4 Gitgraph 20
5 Gitflow 21
5.1 Fork 21
5.2 Parallel collaboration 22
5.3 Pull Request 22
6 Extras 24
6.1 Own project 24
6.2 Learn Git Branching 25
Bibliography 26
7 Appendix 27
A GIT commands 27
AA Review changes and make a commit transaction. 27

AB Synchronize changes 27
B Most used Git commands 29
BA Start a working area 29
BB Work on the current change 29
BC Examine the history and state 29
BD Grow, mark and tweak your common history 29
BE Collaborate 29

1 Goals

In this lab we will learn the basic principles of version control `git` [1]. You must have already done part A of the lab at home to be able to start with part B.

In Section 2 we learn the basic operations to be able to work with Git. The created repository will then be published on GitHub. The advanced functions `branch` and `merge` are tried out in an example in Section 3. In Section 5 we all work together on a repository. Finally, there is some optional work in Section 6.

> ⚠ At the end of the lab, make sure you have published all changes on GitHub. Only the changes published on GitHub will be evaluated.

1.1 Tools

In this lab you will use **Sublime Merge** as a graphical tool or **Git CMD** as a command line.

Figure 1 - Git CMD Commandline

Figure 2 - Sublime Merge GUI

> ⚠ Use the command line process only if you are comfortable with a terminal.
> You will need to know the commands to navigate, display information, etc. `cd, ls, cat, touch, nano ...`

1.1.1 Overview

The interface of Sublime Merge is presented in the Figure 3:

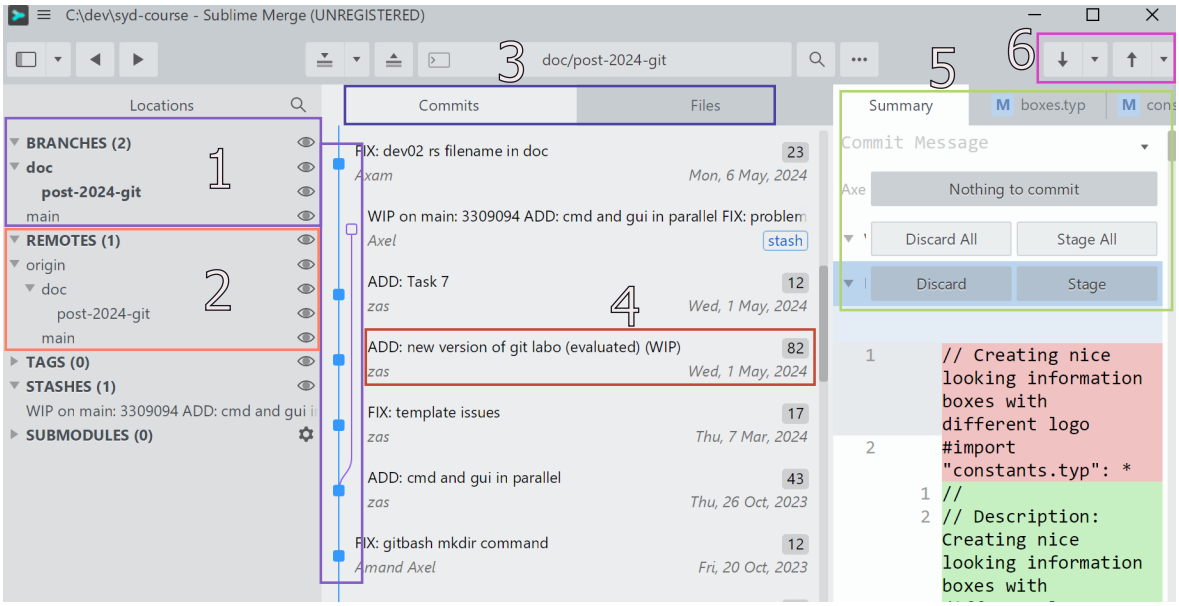


Figure 3 - Sublime Merge GUI

1. **Branches**: list of branches of the repository. You can create, delete and rename branches. They are also displayed in their timeline under the **Commits** tab (point 3).
2. **Remotes**: list of remote repositories, i.e. the servers on which the code is stored and to which you can **push**.
3. **Commits/Files**: tabs to switch between the chronological view of commits and the view of changes for the selected commit.
4. **Commit description**: a commit consists of a message, an author and a date.
5. **Current changes**: the files that have been modified compared to the last commit are listed here. You can select the files you want to **commit** by selecting them and clicking on the **Stage** button. As long as the files have not been **committed**, you can also remove a file from staging by clicking on the **Unstage** button or completely delete a file's changes by pressing **Discard** (⚠️ permanent deletion ⚠️).
6. **Pull / Push**: the two buttons allow you to **pull** - take the latest changes from the remote repository - and **push** - send local changes to the remote repository. It is also possible to **fetch** by clicking on the small arrow next to the **Pull** button, which allows you to see the new commits without modifying the local repository.

2 Basic Operations

The answers to the questions **must** be written down in the Markdown file `answer.md`. The file is located in the repository that will be created in the next step.

2.1 Creating a git repository

In order to work on a common basis, we will create a copy of the template repository. This operation is called `fork` - *Figure 4* . This fork creates a full copy of the original repository which is hosted somewhere else on Github. The changes will then be reflected in this repository, not in the one that was copied. You can therefore work on it without disturbing the work of others.

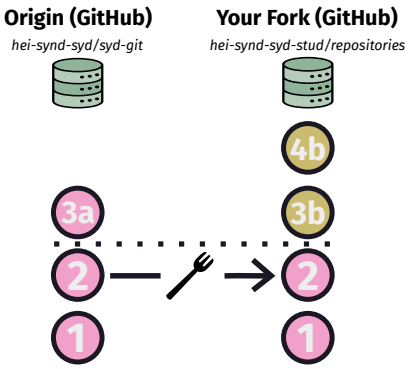
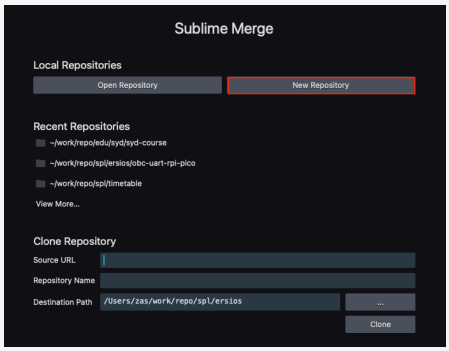


Figure 4 - Fork of the template repository

If you want to create a new repository from scratch (not needed in this labo), you can do so with the following commands or via Sublime Merge GUI:

```
cd /path/to/my/repo
git init my-new-repo
```

`File ⇨ New Repository`



The newly created repository will be empty, without a `remote` which means without a connection to a remote repository (e.g. GitHub). You can add a remote repository later with the command `git remote add origin <url>` or via the GUI.

Use the following link https://classroom.github.com/a/wfORJke1 to automatically create a fork which will be hosted under the GitHub `hei-synd-syd-stud` organization.

For this, you will need to log in to GitHub. Accept the Classroom invitation - *Figure 6* , which will give you the link to YOUR own repository - *Figure 7* .

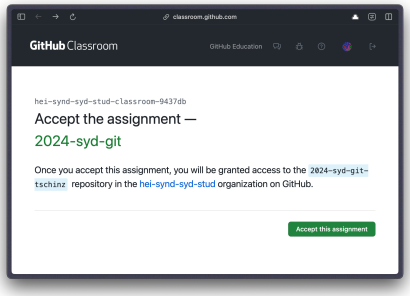


Figure 6 - Invitation link for forking

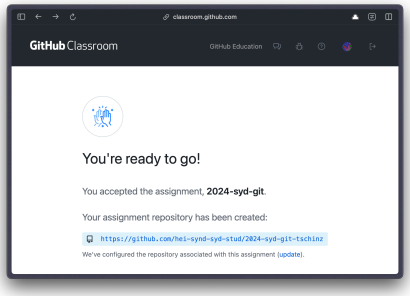


Figure 7 - Link to clone the repository

Note the link to your repository for the next step.
Take the opportunity to visit your repository online at the given link by removing the `.git` at the end of the URL.

2.2 Clone

After you have created the fork, you will receive a link to your own repository - *Figure 7* . Clone it to your local computer at a location of your choice.

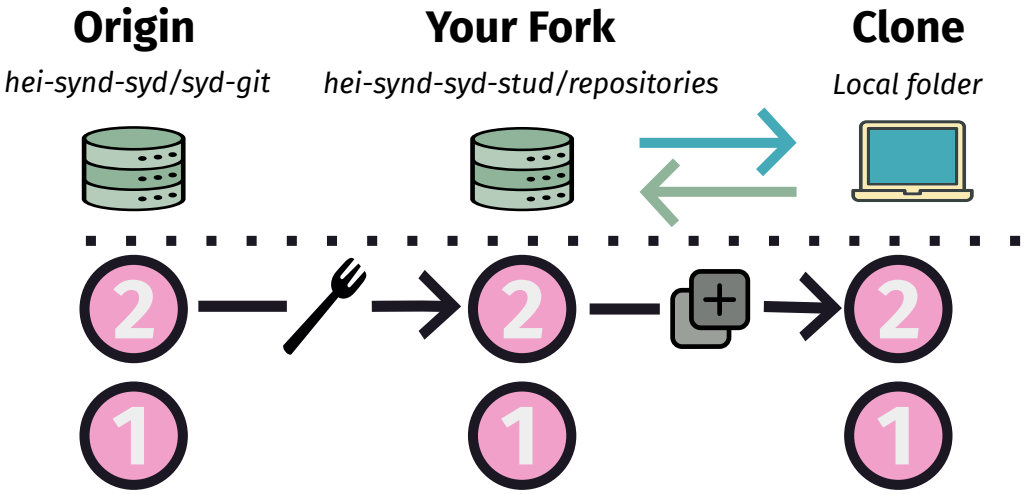


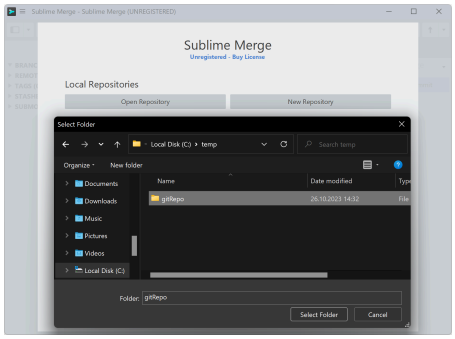
Figure 8 - Cloning

Commandline

```
cd /path/to/my/repo
git clone <linkurl>.git

# Example
git clone https://github.com/hei-synd-syd-stud/
2025-syd-git-tschinz.git
```

GUI - `CTRL+T` ⇨ Paste `Source URL` ⇨ `Select Folder`



2.3 Get status

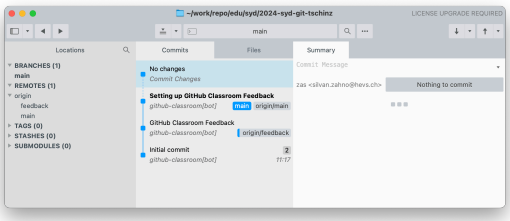
Get information about your repo:

Commandline

```
git status
git log --oneline
```

GUI

See the status in the main window.



Currently, your repository has no changes since nothing has been modified. In Sublime Merge, at the top of the window, it says `No changes`. On the right side, you see `Nothing to commit` meaning we have a copy of the remote repository with no changes added to it.

2.4 Stage & commit

We will now modify data in the repository, in this case the file `answers.md`.

Task 0

Open the document `answers.md` and replace:

- `[students-firstname]` with your first name
- `[students-lastname]` with your last name
- `[github-username]` with your GitHub username

REMOVE THE "[]" from your names.

In the following sections, the **Exercises** must be answered in this same file `answers.md`.

Always **SAVE** your file after answering a task.

Like shown under Section 2.3, check the status of your repo.

Task 1

What is the status of the repository / what do you see different? What does it mean?

We are going to save the changes locally. To do this, it is necessary to `Commit` the changes.

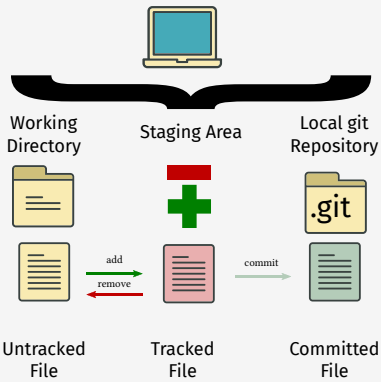


Figure 9 - Types of local Git operations

Your local git repo consists of three areas that are maintained by git:

- Working directory is a directory that contains the current version of your files (a normal file directory in the eyes of your operating system).
- Stage contains the changes to be included in the next commit.
- Head points to the location in the Git repo tree where the next commit should be made.

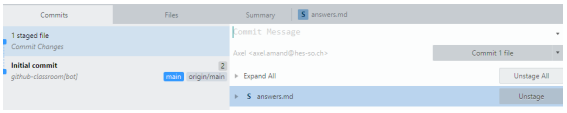
To save the changes *locally*, you first need to select the changes that will be saved in the repo. This process is called `Stage`.

Commandline

```
git add answers.md
```

GUI

Select the file in the upper-right corner, and click on `Stage`.



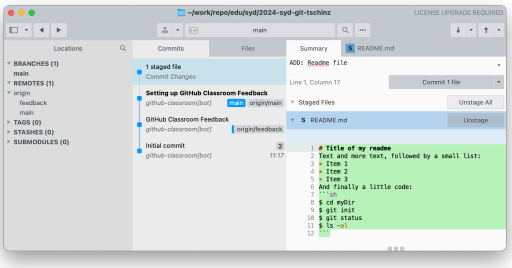
Once all changes are selected, the save is done through a `Commit`. A commit is a snapshot of your repo at a given time and once done, the state of the folder is saved in the repository. The `Commit` is identified by a unique identifier (hash) and contains information about the author, date and commit message, which should reflect the changes made:

Commandline

```
git commit -m "CHG: personal data in
answers.md"
```

GUI

`Commit Message` ⇨ `CHG: personal data in answers.md` ⇨ `Commit 1 file`



Commit your changes

Task 2

After you have made a commit as explained above, answer the following questions:

- Why is the example commit message worded this way: `CHG: personal data in answers.md` ?
- Is it possible to create a commit without a message?
- What is the status of the repo now?
- Is the remote repo (here Github) up to date with our changes?

2.5 Add files

Then create an empty file named `README.md` in the main directory of your repository:

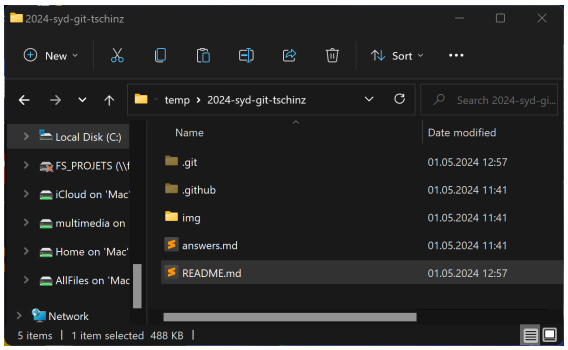
Commandline

```
touch README.md
```

GUI

(Windows) `C:\path\to\repo\` ⇨ `New` ⇨ `Any File` ⇨ `README.md`

(Linux&MacOS) `/path/to/repo` ⇨ `New` ⇨ `Any File` ⇨ `README.md`



Task 3
Have another look at the status of your repo, what do you see?

Make a commit with the new file and the answers to exercises 2 and 3.

Example repository

A simple Git repo consisting of five commits can be represented as follows. The `Head` position is a reference to a commit that represents the current state/view of the repo, in this case the latest change:

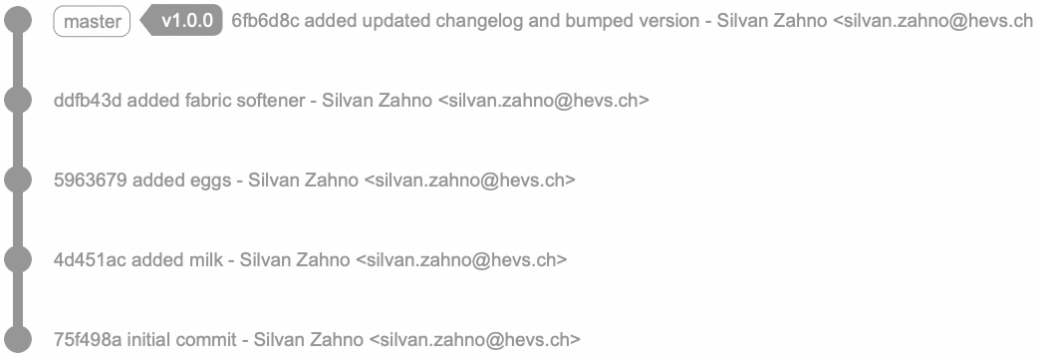
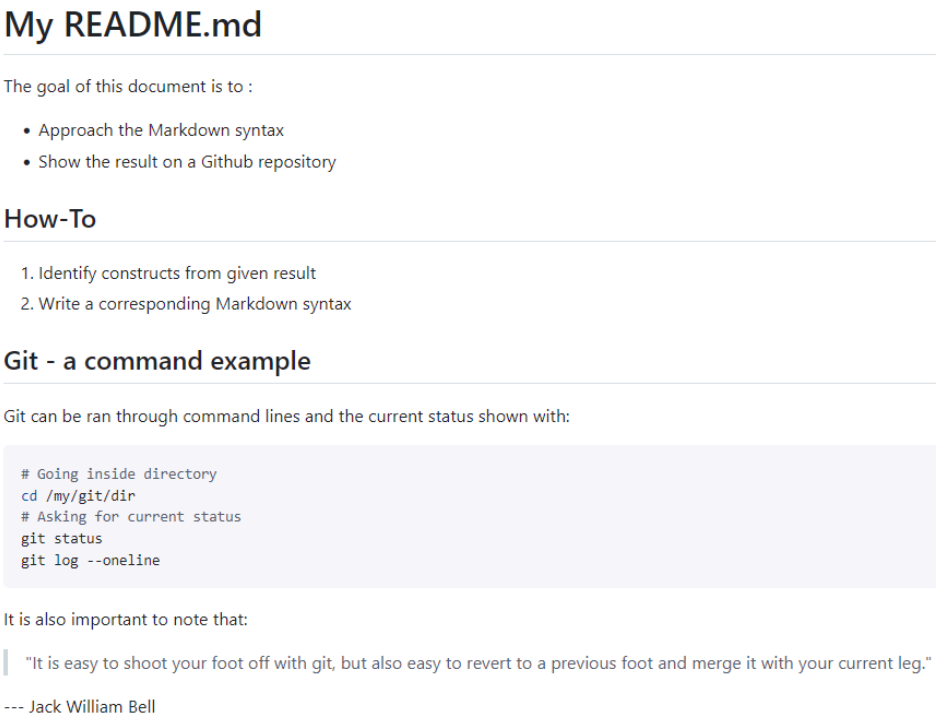


Figure 10 - Five commits on the local repo, each commit has its own identification

2.6 Markdown

Modify the `README.md` file with a text editor to make it resemble roughly like the following:



My README.md

The goal of this document is to :

- Approach the Markdown syntax
- Show the result on a Github repository

How-To

1. Identify constructs from given result
2. Write a corresponding Markdown syntax

Git - a command example

Git can be ran through command lines and the current status shown with:

```
# Going inside directory
cd /my/git/dir
# Asking for current status
git status
git log --oneline
```

It is also important to note that:

> "It is easy to shoot your foot off with git, but also easy to revert to a previous foot and merge it with your current leg."

--- Jack William Bell

Figure 11 - Markdown structure to reproduce

Use a tool as shown in chapter `Labo Part A - Markdown` to help you write and preview your markdown file.

Here is a Markdown cheatsheet reminder to help you create your `README.md` file:

```
# Title
## Subtitle
### Sub-subtitle

Some text

Some code:
```bash
cd myDir
```
```

```
A list:
  * Item 1
  * Item 2

> Quoting something

A numbered list:
  1. Item 1
  1. Item 2
```

Complete your `README.md` file.

Once satisfied, commit your changes.

2.7 Push online

Currently, all the work done is only saved *locally*. To put it online, it is necessary to `push` the commits to the remote repository (Github) in order to have a copy of it and, as part of group work, allow others to see it:

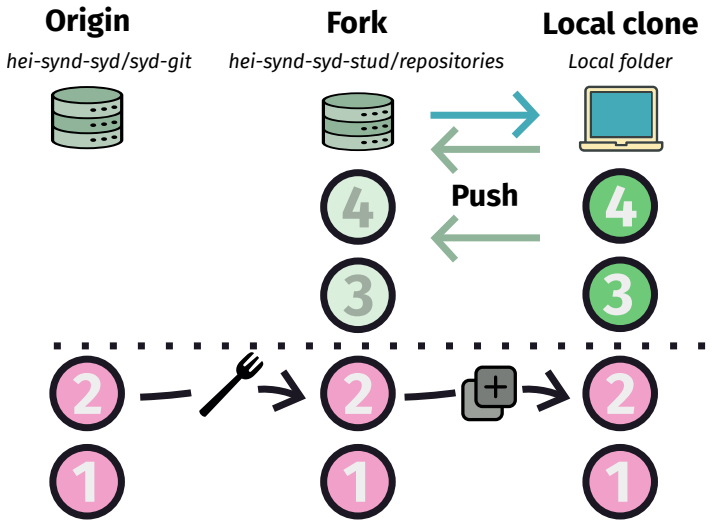


Figure 12 - Push to the remote repository

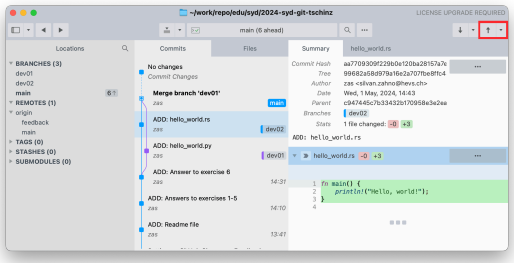
To do this, once all changes have been committed:

Commandline

```
git push origin main
```

GUI

`Top Right Arrow` ⇨ `Push changes`



Go online to Github to see the result of your work.
You should see the `README.md` file with the content you created in readable form.

2.8 Back in time (not the future)

Git is a version control system. It is therefore possible to return to an earlier version of the repository. To do this, you first need to identify the commit you want to go back to. Here we go back to the very first commit.

Every commit is provided with a "hash" or "checksum" (of type `sha1`). This hash is a unique identifier that allows you to isolate a specific commit. The latter is visible by clicking on a commit for SublimeMerge, or by using the following command:

```
git log --oneline
```

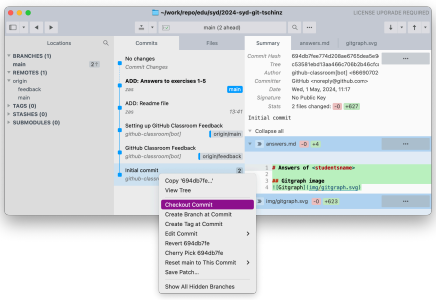
To change commits, perform the `checkout` operation:

Commandline

```
git checkout <SHA1>
# for example
git checkout 694db7f
```

GUI

`Select First Commit` ⇨ `Right click` ⇨ `Checkout commit`



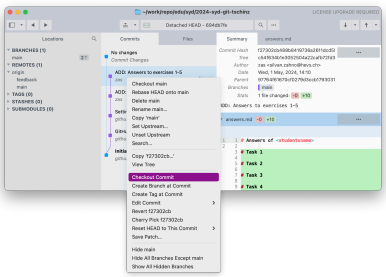
Then go back to the last commit in the same way. Apart from the hash, it is possible to return to the last commit by using the name of the branch, here `main`.

Commandline

```
git checkout main
```

GUI

`Branches (1)` ⇨ `main` ⇨ `checkout main`



Task 4

What happens in your working directory when going back to the "Initial commit" commit? What happens when going back to the last commit?

2.9 General questions

Task 5
What is the difference between the local repository and the remote repository?
What would happen if you deleted the local repository?

Task 6
What about the original repository that was forked?
Has it been modified?

Make a commit containing the previous answers.

Push your changes to the remote repository on Github before continuing.

2.10 Tag

To identify key commits, it is possible to create `tags`. A tag is a pointer to a particular commit. It is often used to mark versions of code, for example `v1.0`, `v2.0`, etc.

To mark the end of this chapter, create a tag `chapter2` on the last commit. To do this:

Commandline

```
git tag chapter2
git push origin chapter2
```

GUI

- `Right click on commit` ⇨ `Create Tag at Commit` ⇨ `chapter2` ⇨ `Enter`
- `Right click on tag` ⇨ `Tag chapter2` ⇨ `Push Tag chapter2`

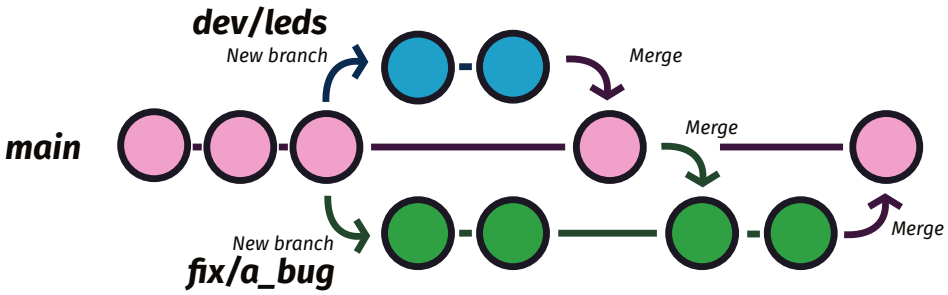
Check online on Github that the tag `chapter2` has been successfully created and pushed, along with your latest changes.

This chapter is to be completed in the same repository as before.

Make sure you have committed and pushed all your changes before starting this chapter, and that these have been tagged by `chapter2`.

3 Branch and Merge

So far, we have used the basic functions of git - pink commit line.
There are also the `branches` and `merge` functions, which Git has greatly simplified compared to previous tools:



Branches allow you to work in parallel on multiple versions of the same project. For example, you can create a branch to develop a new feature, while your colleague works on a patch to fix a bug. The merge action (`merge`) allows you to combine changes made on one branch into another.

Git is not omniscient. If two branches have modified the same files, Git will not know how to merge them, and you will have to manually resolve the conflicts.

3.1 First Branch

The first branch simulates the work of a first person on a new file, `hello_world.py`. This could be any file: code, image, Inventor drawing, Matlab simulation ...

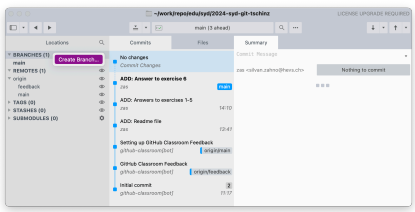
1. Create a development branch `dev01` in your local repo:

Commandline

```
git checkout -b dev01
```

GUI

`Branches` ⇨ `Create Branch` ⇨ `dev01`



2. Create a commit on this branch:

- Create a file `hello_world.py`
- Fill it with the following code:

```python
print("Hello, world!")
```

Make a commit as presented in chapter Section 2.4, with a clear message.

3.2 Second Branch

The second branch simulates the work of a second person on another file, `hello_world.rs`.

Assuming the branch was created at the same time as the other, it is necessary to first return to the same commit as before.

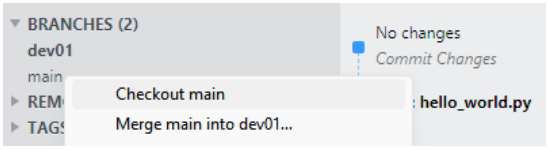
1. Switch back to the `main` branch:

Commandline

```
git checkout main
```

GUI

`main` ⇨ `Right click` ⇨ `Checkout main`



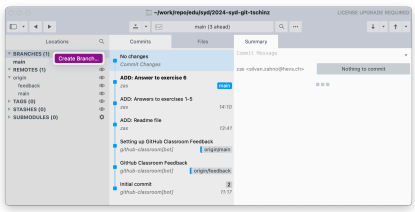
2. From the main branch, create a new development branch `dev02`:

Commandline

```
git checkout -b dev02
```

GUI

`Branches` ⇨ `Create Branch` ⇨ `dev02`



3. Create a commit on this branch:
   - Create a file `hello_world.rs`
   - Fill it with the following code:

```rust
fn main() {
  println!("Hello, world!");
}
```

Make a commit as presented in chapter Section 2.4, with a clear message.

3.3 Merge dev02

Now that both branches have received commits, it is time to merge them into the main branch `main`.

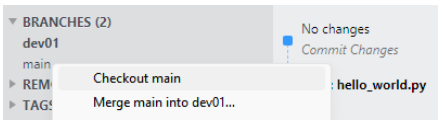
1. Switch back to the `main` branch:

Commandline

```
git checkout main
```

GUI

`main` ⇨ `Right click` ⇨ `Checkout main`



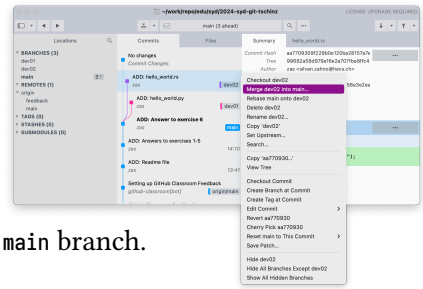
2. Merge the `dev02` branch into `main`:

Commandline

```
git merge dev02
```

GUI

Select `Commit` ⇨ `Merge dev02 into main ...`



The work of the `dev02` branch is now integrated into the `main` branch.

3.4 Merge dev01

Similarly, it is now time to merge the `dev01` branch into `main`.

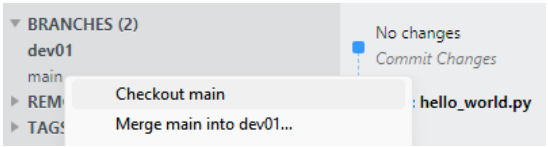
1. Ensure you are on the `main` branch:

Commandline

```
git checkout main
```

GUI

`main` ⇨ `Right click` ⇨ `Checkout main`



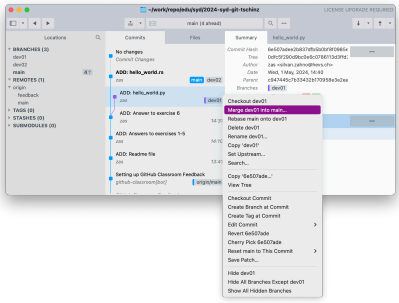
2. Merge the `dev01` branch into `main`:

Commandline

```
git merge dev01
```

GUI

Select `Commit` ⇨ `Merge dev01 into main ...`



3.5 Final Result

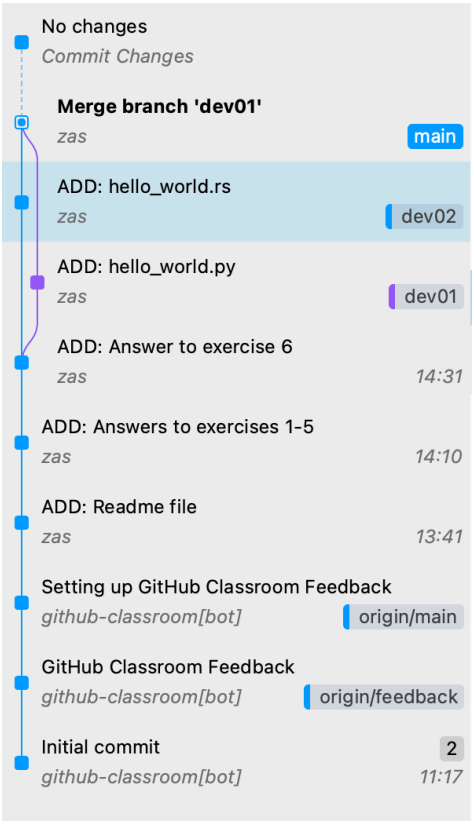


Figure 22 - Status after repo branches `dev01` and `dev02` have been merged

By default, the repository might only display a blue line. Since the work is very linear, Sublime Merge hides the commits. You can force the display of all commits by clicking on the button  on the commit lifeline.

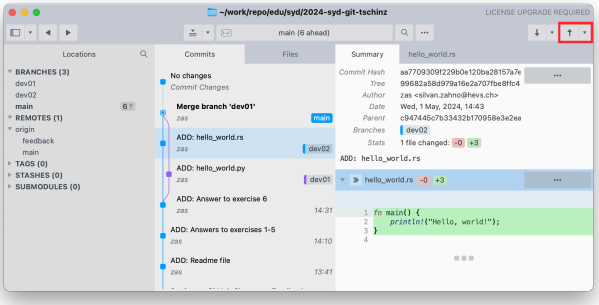
1. Push your repository to the remote GitHub server. **Three `push` commands are required to push each branch separately**:

Commandline

```
git push origin main
git push origin dev01
git push origin dev02
```

GUI

`Checkout dev01` ⇨ `Top Right Arrow` ⇨ `Push changes`
`Checkout dev02` ⇨ `Top Right Arrow` ⇨ `Push changes`
`Checkout main` ⇨ `Top Right Arrow` ⇨ `Push changes`



2. Tag the `main` branch with the tag `chapter3` as presented in chapter Section 2.10. Don't forget to push the tag!

Check online on Github that the tag `chapter3` has been successfully created and pushed, along with your latest changes.

The status of your repository should be similar to Figure 22 in addtions to the tags `chapter2` and `chapter3`. This is part of the evaluation.

This chapter is to be completed in the same repository as before.

Make sure you have committed and pushed all your changes before starting this chapter, and that these have been tagged by `chapter3`.

4 Gitgraph

The Figure 24 shows an example of a git repository. Name all the elements that can be seen in the image (points 1-10).

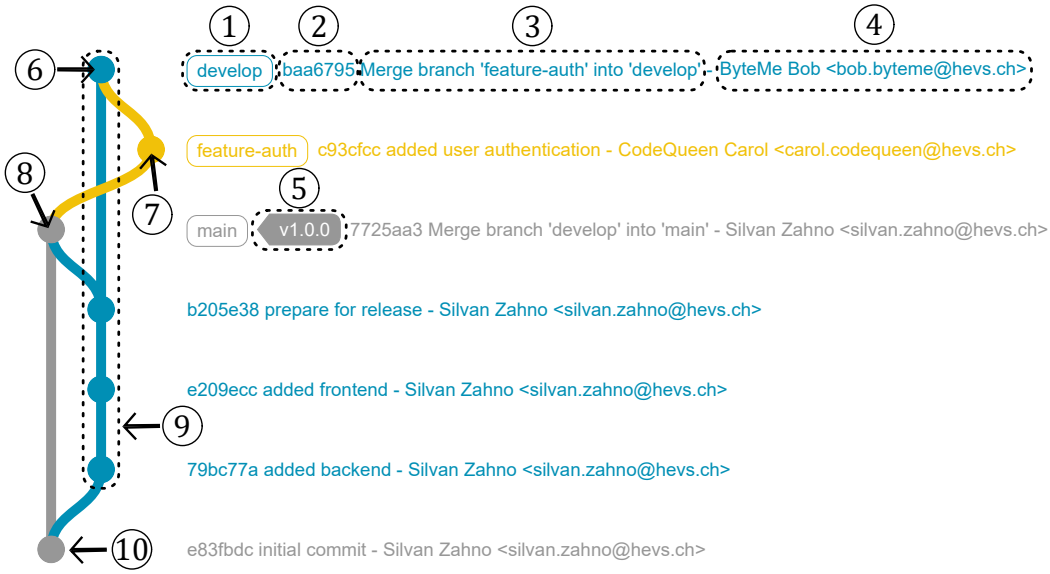


Figure 24 - Example of a Gitgraph

Task 7

1. Name all elements that can be seen in the image Figure 24 (points 1-10)
2. 
   Commit your changes
3. Push your local repository to your cloud GitHub repository
4. Tag and push the tag `chapter4` according to Section 2.10

Check online on Github that the tag `chapter4` has been successfully created and pushed, along with your latest changes.
Congratulations, the work on your first GitHub repository is finished.

*This chapter is to be completed in a **different repository** than before.*

Make sure you have committed all your changes before starting this chapter, and that these have been tagged by `chapter4` and pushed online. Changes, commits, tags ... that are not available online will count as unfinished work for the grading of the lab.
You can check online under Github if your repository is available and up to date.

5 Gitflow

For this task, use the Gitflow philosophy presented in the course. You will all collaborate on the following Git repo as if you were forming a development team:

https://github.com/hei-synd-syd/2026-syd-gitflow [2]

This is a public Git repo hosted on Github.

5.1 Fork

For security reasons, you are not allowed to work directly on this repository. You need to create your own copy (`fork`) in order to make changes. Therefore, please create a `fork` of this repository in your GitHub account. To do this, use the `fork` button in the Github web interface.

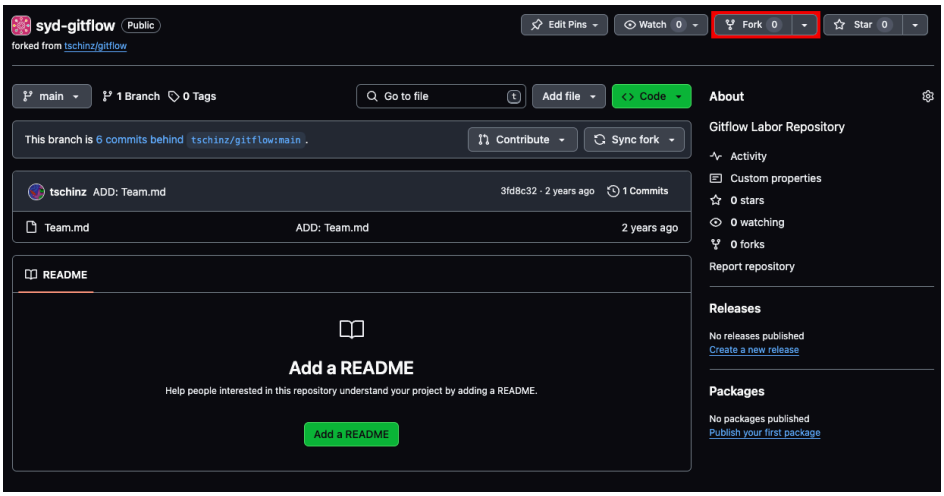


Figure 25 - Fork button for a GitHub repository

Then clone this forked repo. The URL of your new repo will look like this:

```
git clone https://github.com/<username>/<year>-syd-gitflow.git
```

5.2 Parallel collaboration

You will all edit the same file. To avoid conflicts, edit **only the line that is relevant to you**.

Ask the professor or teaching assistant to know your line number!

Edit the `Team.md` file.
Answer the question based on the number you received.

`Commit` and `push` your branch to your fork of the repository on GitHub.

5.3 Pull Request

Now that your changes are ready, it is still necessary to integrate them into the original repository https://github.com/hei-synd-syd/2026-syd-gitflow.
Since you do not have the rights to work on it directly, the only possibility is to make a merge request - `pull request` - on the original repository.
Thus, the administrator can accept the changes, ask you to correct items before accepting the merge, discuss issues, etc.
To do this, use the interface of the GitHub website ⇨ `Contribute` ⇨ `Open pull request`.

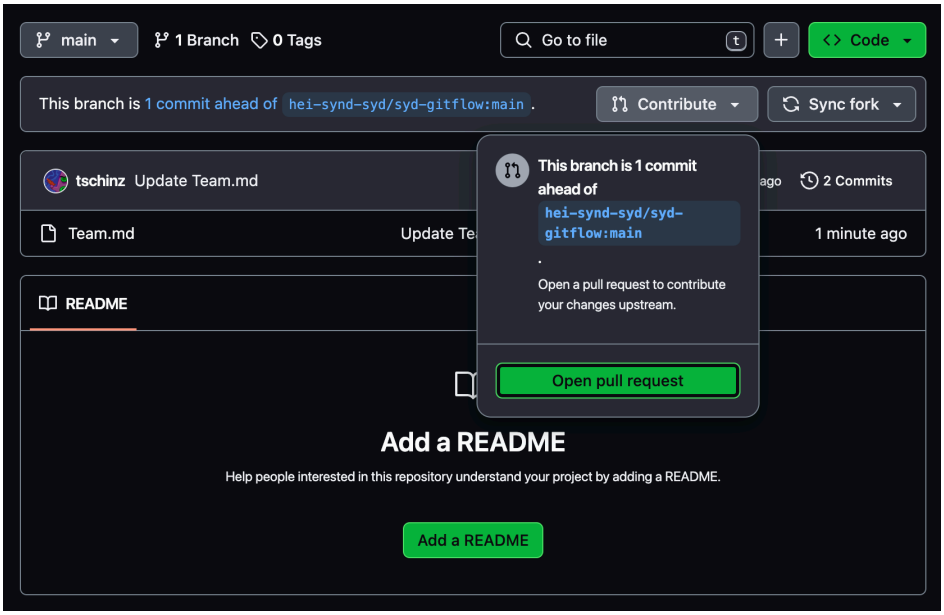


Figure 26 - Create Pull Request on Github

You will need to give a meaningful title and a short description to your pull request. It is possible to write the description in Markdown - Section 2.6.

The incomplete, poorly described, poorly titled pull requests … will be rejected.

When the pull requests are ready, the merges are done by the teacher and assistants.

Check online under https://github.com/hei-synd-syd/2026-syd-gitflow/pulls that:

- Your pull request has been correctly **opened**
- It contains a clear **title**
- It contains a clear **description**

You can still update your pull request description if necessary until it is accepted / rejected.

Missing pull requests will count as uncompleted work for the lab grading.

6 Extras

This optional chapter can be started provided the previous tasks have been completed. There are 2 tasks to do:

1. Put your own project on Github and provide it with a `README.md` and a CI/CD.
2. Follow the “Learn Git Branching” website

6.1 Own project

- Put a current project you are working on, on github.
- Create a `README.md` file for the project using the Markdown Syntax. The `README.md` should include the following:
  - Title
  - Image
  - Description of the project
  - Explanation how to run / use the project
  - List of authors
- Now create a github action to turn the `README.md` into a PDF on every push. For this find a suitable github action and add it to your project.

If you need help to create the github actions. Checkout the following hint.

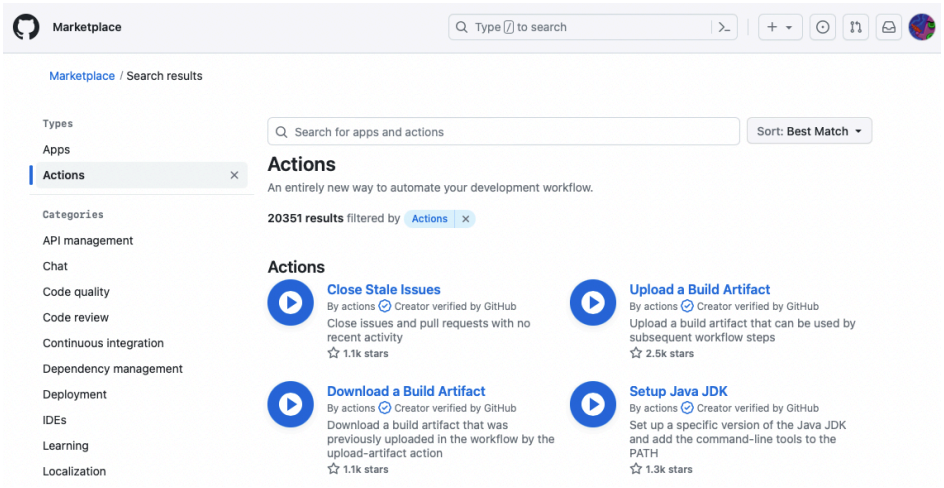


Figure 27 - Github Action Marketplace

6.2 Learn Git Branching

Follow the Tutorial on https://learngitbranching.js.org.

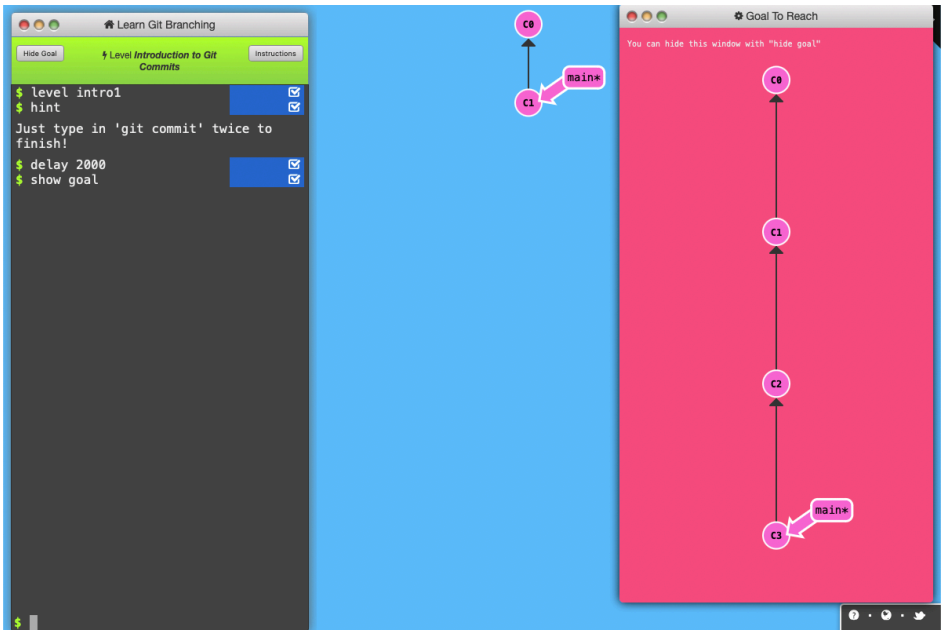


Figure 28 - Learn git branching website

Bibliography

[1] T. Linus, “Git.” Accessed: Apr. 25, 2023. [Online]. Available: https://git-scm.com/

[2] tschinz, “Tschinz/Gitflow.” Accessed: Apr. 25, 2023. [Online]. Available: https://github.com/tschinz/gitflow

[3] gitlab, “Git Cheatsheet.” 2023.

[4] “GitHub Git Spickzettel.” Accessed: Apr. 25, 2023. [Online]. Available: https://training.github.com/downloads/de/github-git-cheat-sheet/

7 Appendix

A GIT commands

Github git cheatsheet [3], [4]

AA Review changes and make a commit transaction.

```
git status
```

Lists all new or changed files ready for commit.

```
git diff
```

Displays file changes that have not yet been indexed.

```
git add [file]
```

Indexes the current state of the file for versioning.

```
git diff --staged
```

Shows the differences between the index (“staging area”) and the current file version.

```
git reset [file]
```

Takes the file from the index, but preserves its contents.

```
git commit -m "[descriptive message]"
```

Adds all currently indexed files permanently to the version history.

AB Synchronize changes

Register an external repository (URL) and swap the repository history.

```
git fetch [remote]
```

Downloads the entire history of an external repository.

```
git merge [remote]/[branch]
```

Integrates the external branch with the current locally checked out branch.

```
git push [remote] [branch]
```

Pushes all commits on the local branch to GitHub.

```
git pull
```

Pulls the history from the external repository and integrates the changes.

B Most used Git commands

BA Start a working area

- `clone` - Clone a repository into a new directory
- `init` - Create an empty Git repository or reinitialize an existing one

BB Work on the current change

- `add` - Add file contents to the index
- `mv` - Move or rename a file, a directory, or a symlink
- `reset` - Reset current HEAD to the specified state
- `rm` - Remove files from the working tree and from the index

BC Examine the history and state

- `log` - Show commit logs
- `show` - Show various types of objects
- `status` - Show the working tree status

BD Grow, mark and tweak your common history

- `branch` - List, create, or delete branches
- `checkout` - Switch branches or restore working tree files
- `commit` - Record changes to the repository
- `diff` - Show changes between commits, commit and working tree, etc
- `merge` - Join two or more development histories together
- `rebase` - Reapply commits on top of another base tip
- `tag` - Create, list, delete or verify a tag object signed with GPG

BE Collaborate

- `fetch` - Download objects and refs from another repository
- `pull` - Fetch from and integrate with another repository or a local branch
- `push` - Update remote refs along with associated objects