Introduction to git - Part A & B

Contents

1 Goal

This lab is divided into two parts. Part A must be done at home as preparation, while Part B is done together in the lab. The lab will be done independently on your laptop and graded at the end.
In this lab we will learn the basic principles of version control `git` [1], especially the tools Git command line and Sublime Merge, which need to be installed and configured on your machine (see Section 2). Furthermore, accounts are created on the platforms Github and Hevs Gitlab [2].
Finally, we will learn the basics of Markdown in Section 3 to easily write text files.

It is crucial that the installation and configuration is done carefully to avoid wasting time during part B of the laboratory.

2 Installation

The first step is to install Git as well as Sublimemerge. You can choose whether you want to use the command line or the GUI during the lab. However, both tools should be installed and configured.

2.1 git

You can download the latest version from the official website https://git-scm.com/ [1]. Git is available for Linux, Mac, and Windows. This lab requires git ≥ 2.27.

2.1.1 Command line

Start "Git Bash" on Windows or "Terminal" on MacOS. This is a Unix/Linux-like command editor that allows you to run Git commands in console mode.

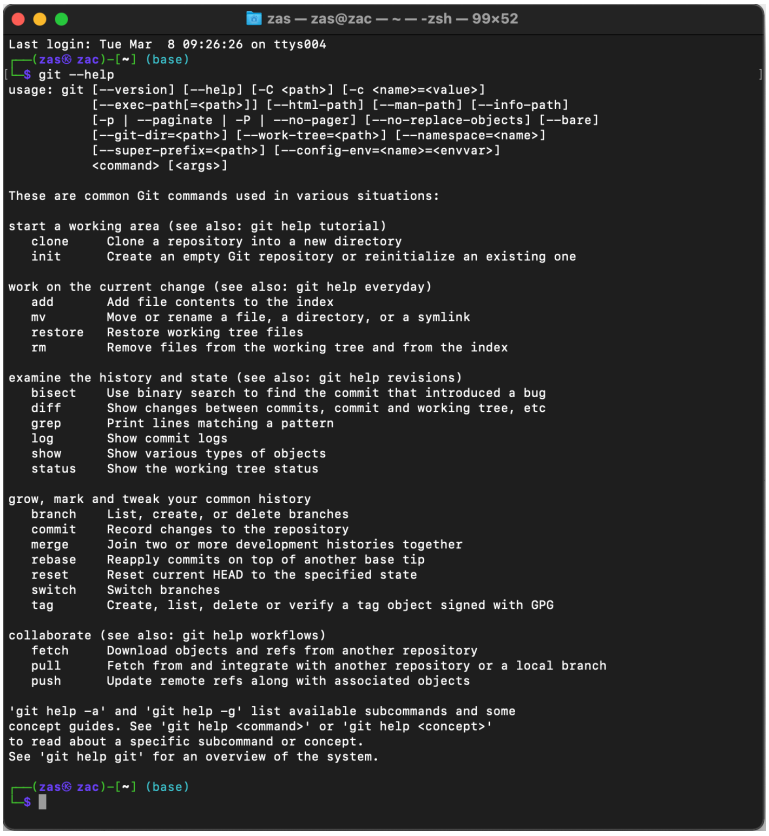


Figure 1: git Terminal

Note that for all commands in Git Bash, you can get help by inserting `--help` after the command.

```
git --help
```

2.1.2 Global configuration

A variety of settings can be configured in Git. It is possible to change the settings globally on your computer (flag `--global`) or only for a specific repository.
We will now perform the minimal configuration. Use the following commands to set your identity in Git globally on the system. Use your name and email address. This information is publicly visible to identify your work (your commits).

```
git config --global user.name "Firstname Lastname"
git config --global user.email first.last@email.ch
```

For example:

```
git config --global user.name "Silvan Zahno"
git config --global user.email silvan.zahno@hevs.ch
```

You can check the configuration with the following command:

```
git config --list
```

You can also check a specific setting:

```
git config user.name
```

2.2 Sublime Merge

Visit the website https://www.sublimemerge.com and download and install the Sublime Merge tool [3].

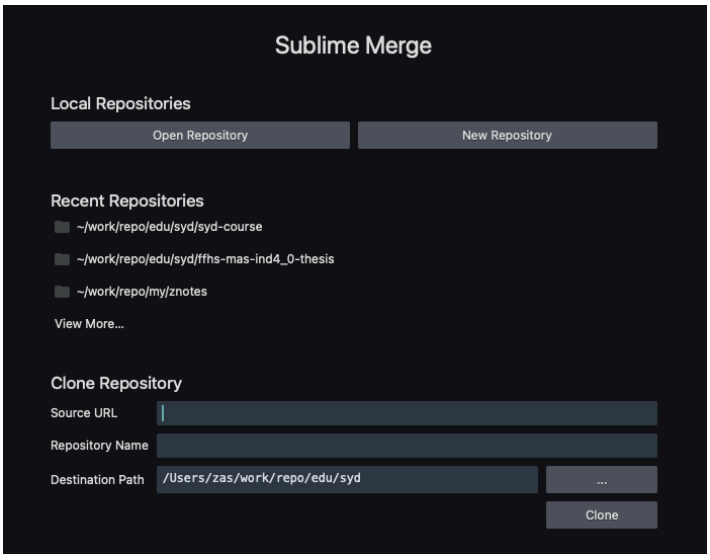


Figure 2: Sublime Merge GUI

2.3 Online accounts

2.3.1 Gitlab

Visit the website https://gitlab.hevs.ch and log in with your school account (SwitchEDU-ID).

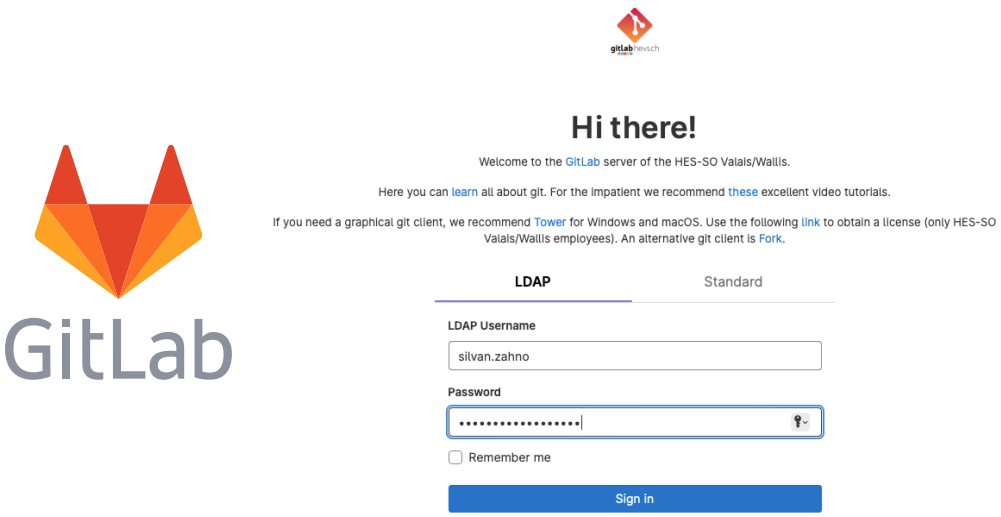


Figure 3: Gitlab Login

2.3.2 Github

Visit the website https://github.com and create an account and log in.

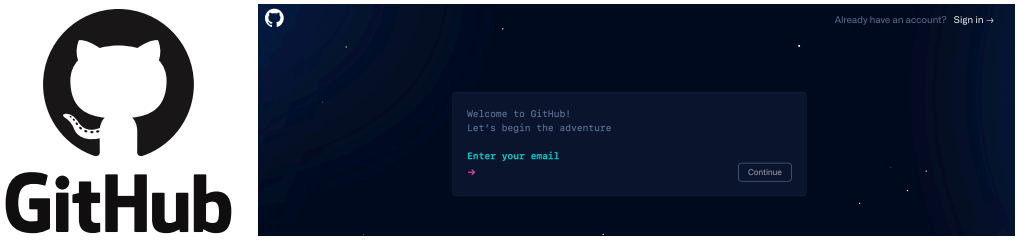


Figure 4: GitHub Login

2.4 Windows Configuration

In order to see also the hidden `.git/` folder as well as file extentions. Configure your Windows File Explorer as follows Figure 5:
File Explorer ⇨ View ⇨ Show ⇨ Activate "**File name extensions**" and "**Hidden items**"

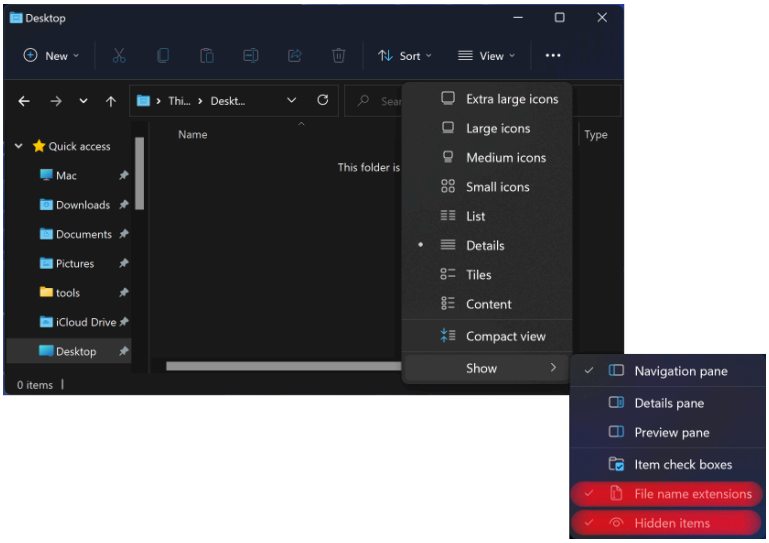


Figure 5: Windows File Explorer Configuration

3 Markdown

Markdown is a lightweight markup language with plain-text formatting syntax. It is designed to be easy to read and write, while also being easily converted into PDF, HTML or other formats. Markdown is commonly used for formatting text on the web, such as in `README.md` files, documentation, forum posts, and messaging.

In order to write Markdown, you need your preferred Text editor or you can install a spezialised Markdown Editor such as Marktext.

For example the intropage of this course is written in Markdown, you can see the source code of the page by clicking on the "Edit this page" button on the top right corner of the page or via the link.

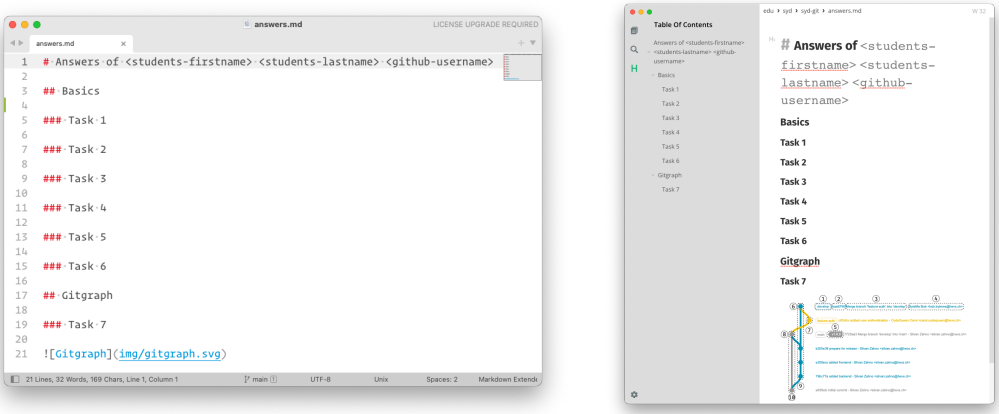


Figure 6: Left: Simple Texteditor (Sublime Text), Right: Marktext

For the lab, you will need to write a documents in Markdown format. Be ready with your editor.

3.1 Markdown syntax

Hereafter a short overview about how a markdown file is structured. The syntax is simple and easy to learn. The file has to be saved with the extension `.md`. A more complete syntax list can be found at here.

```
# Title 1

## Title 2

### Title 3

Some simple Text _italic_ **bold**
~~Strikethough~~ `monospaced`

Fomulas $S = \sum_{i=1}^{n} x_{i}^{2}$

- List Item 1
- List Item 2
1. Numbered List Item 1
2. Numbered List Item 2

[Link name](https://hevs.ch/synd)

![logo](logo.svg)

```rust
// A rust code bloc
fn main(){
  println!("Hello World");
}
```

Tables	Are	Cool
col 1	left-align	f_{clk}
col 2	centered	$12
col 3	right-align	1024

```

4 Outro

Congratulations You have now installed and configured everything you need to work with Git. Prepare for the next lab:

- Study the theory
- Familiarize yourself with the Git commands
- Familiarize yourself with the graphical tool Sublimemerge
- Practice writing documents with Markdown

See the appendix Section A and Section B for a summary of the most important Git commands.

5 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 7 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 8. In Section 10 we all work together on a repository. Finally, there is some optional work in Section 11.

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.

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

6 Tools

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

Figure 7: Git CMD Commandline

Figure 8: Sublime Merge GUI

7 Basis Operationen

7.1 Erstellen eines git Repositories

Erstellen Sie einen Fork des Template Repository mithilfe des folgenden Links https://classroom.github.com/a/O0eniBP2 (Figure 9) Hierfür müssen Sie sich bei GitHub anmelden.

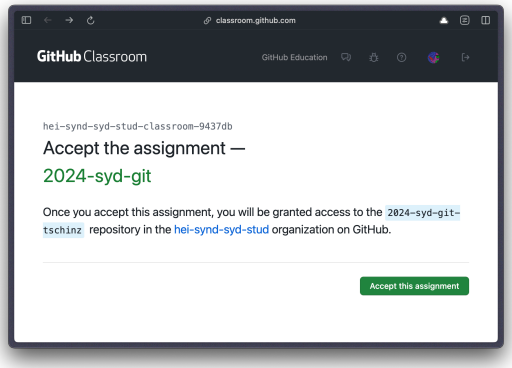


Figure 9: Invitation link for forking

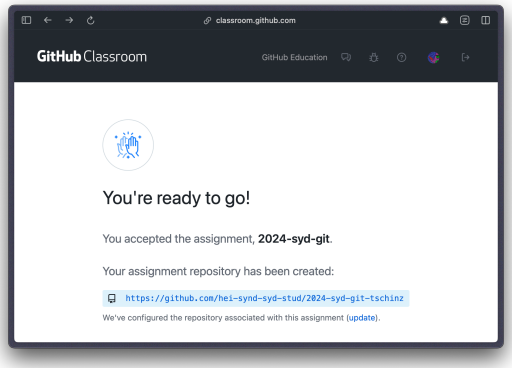


Figure 10: Link to clone the repository

7.2 Clone

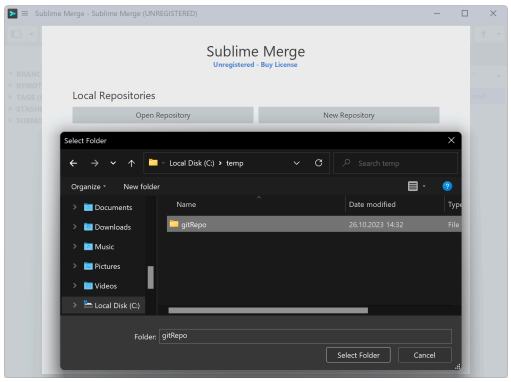
After you have created the fork, you will receive a link to your own repository (Figure 10). Clone it to your local computer at a location of your choice.

Commandline

```
git clone <linkurl>.git
e.g.
git clone https://github.com/hei-synd-syd-stud/2024-syd-git-tschinz.git
```

GUI

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



Task 0

Change the title of the document `answers.md` and replace:

- `<students-firstname>` with your first name
- `<students-lastname>` with your last name
- `<github-username>` with your GitHub username

Task 1

What is in the directory after the Git repo has been cloned? What are the different files and folders good for?
Write down the answers in the file `answers.md`!

7.3 Get status

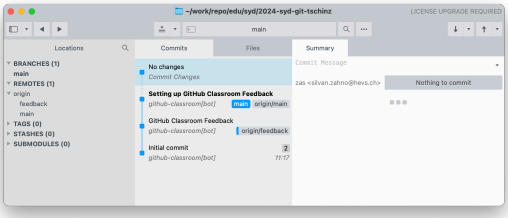
Get information about your repo with the following commands:

Commandline

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

GUI

See the status in the main window.



7.4 Add file

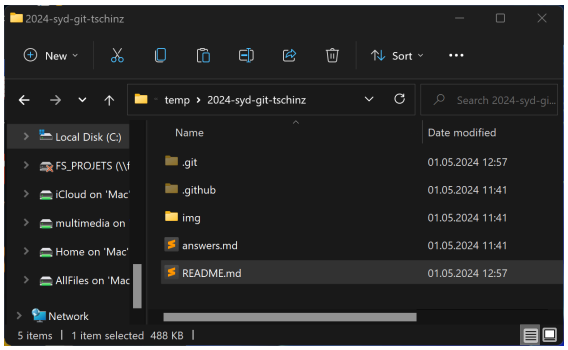
Now create an empty file named `README.md` in the root directory of your repo.

Commandline

```
touch README.md
```

GUI

`C:\[myRepo]` ⇨ `New` ⇨ `Any File` ⇨ `README.md`



Use the previous commands to retrieve the information about your repo again. What has changed?

Task 2
What has changed in `git status` and `git log -oneline`? And why?
Write down the answer

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.

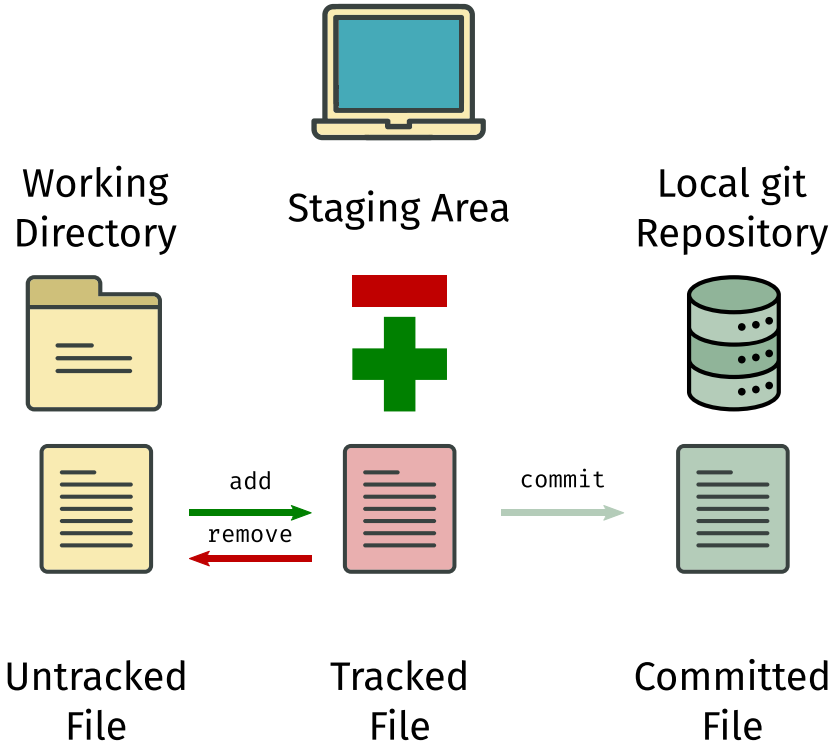


Figure 11: Types of local Git operations

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.

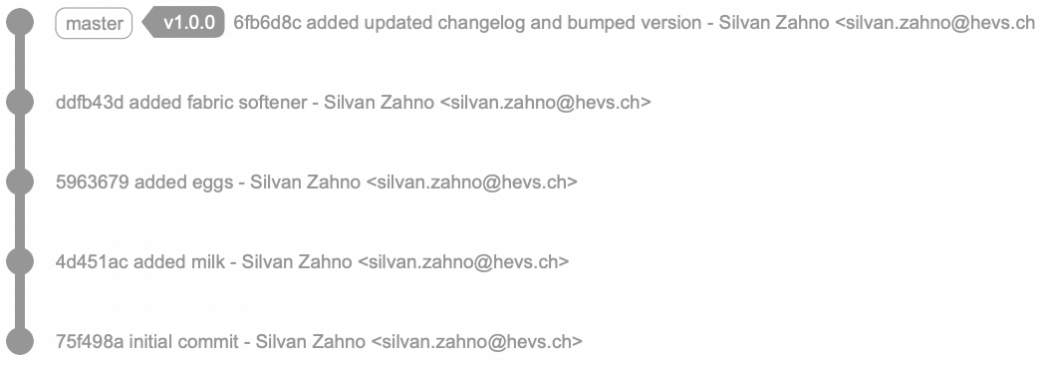


Figure 12: Five commits on the local repo, each commit has its own identification

7.5 Add file to repo

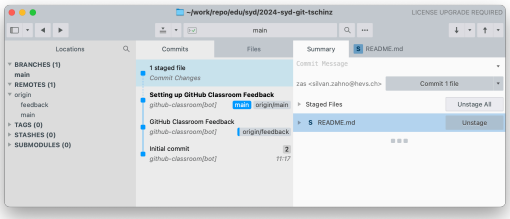
Add the previously created file `README.md` to the stage with the command:

Commandline

```
git add README.md
```

GUI

`Untracked Files` ⇨ `README.md` ⇨ `Stage`



Task 3

Have another look at the `git status` info on your repo, what do you see? Write down the answer!

Edit the file `README.md` with a texteditor and insert the following text (markdown syntax):

````
# Title of my readme
Text and more text, followed by a small list :
  * Item 1
  * Item 2
  * Item 3

And finally a little code:
```sh
$ cd myDir
$ git init
$ git status
$ ls -al
```
````

Task 4

Have another look at the `git status` info on your repo, what do you see? Write down the answer!

7.6 Add new changes

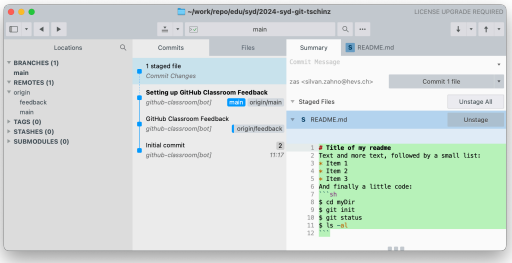
Add the latest version of the `README.md` file to the stage.

Commandline

```
git add README.md
```

GUI

`README.md` ⇨ `Stage`



7.7 Execute commit

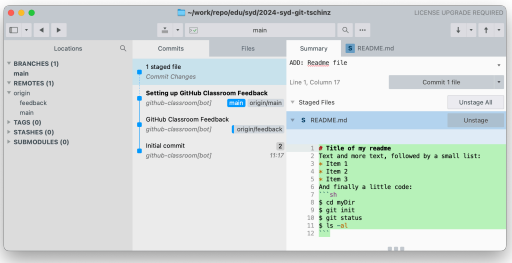
Now perform a commit with the following command:

Commandline

```
git commit -m "ADD: README file."
```

GUI

`Commit Message` ⇨ `ADD: README file.` ⇨ `Commit 1 file`



The `-m` option allows to specify the message of the commit directly. This message must be self-explanatory. It corresponds to the description of the changes. It is possible to insert a text block e.g. via a text editor without using the `-m` option.

Your changes are now published to your local git repo. Bravo.

7.8 More information

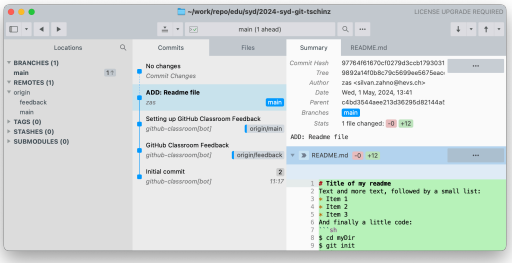
What information can you get now with the command:

Commandline

```
git log --oneline
```

GUI

`Select First Commit` ⇨ `See all informations`



Exercise 5

Explain clearly all the information contained in the first line.

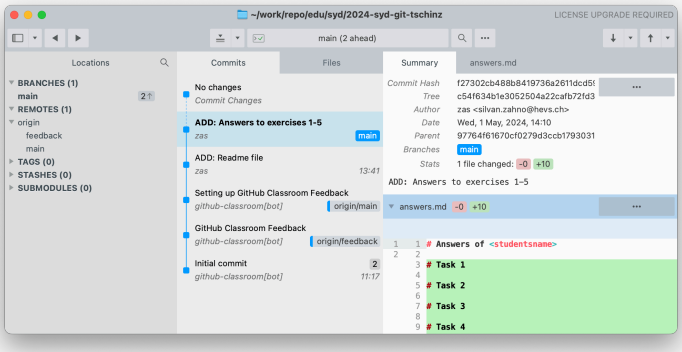
- What does the string at the beginning mean?
- What do `HEAD` and `main` mean?
- What's after the brackets?

Write down the answer!

7.9 Checkout commit

To ensure that your answers are not lost, save the changes to the `answers.md` file in a commit.

```
git add answers.md
git commit -m "ADD: Answers to exercises 1-5"
```



Note that each commit is provided with a “hash” or “checksum” (of type sha1). The hases shown with the command:

```
git log --oneline
```

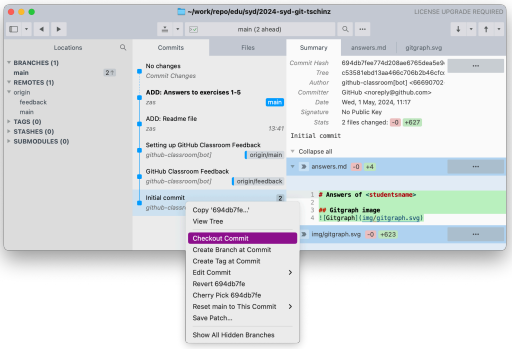
The showns hashes are only the first few characters of these so-called “short hashes”.
Now do a checkout with the following command using the “short hash” , which corresponds to the first commit.

Commandline

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

GUI

`Select First Commit (Initial Commit)` ⇨ `Checkout commit`



Now take a close look at the contents of the folder.

7.10 Checkout master

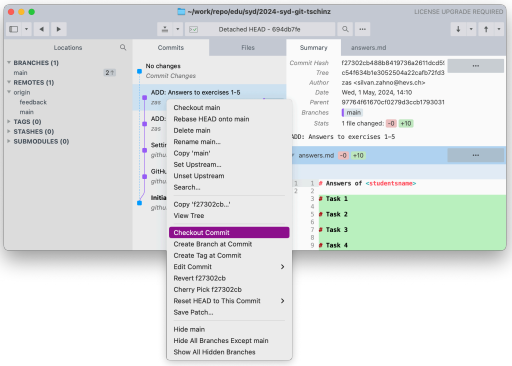
Now run a checkout with the following command:

Commandline

```
git checkout main
```

GUI

`Branches (1)` ⇨ `master` ⇨ `checkout master`

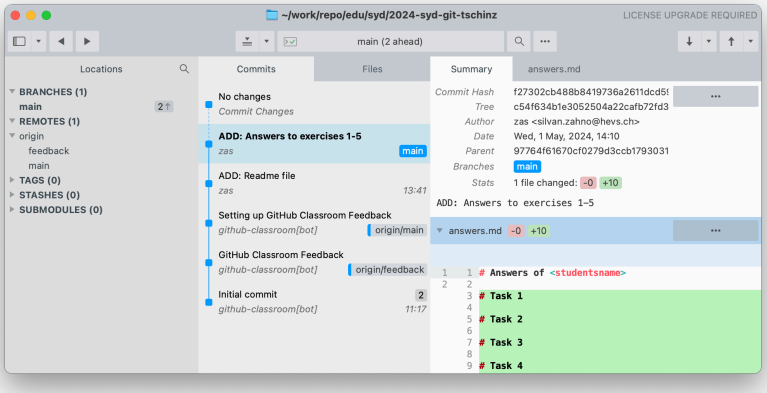


Task 6

What did you notice happened to the files and folders in the Project folder when you took the “initial commit”? What happened when they jumped back to the last commit? Write down the answer!

To ensure that your answers are not lost, save the changes to the `answers.md` file in a commit.

```
git add answers.md
git commit -m "ADD: Answer to exercise 6"
```



8 Branch and Merge

So far we have used the basic functions of Git. There are also the branch and merge functions, which Git has greatly simplified compared to the tools that existed earlier.
For this hands-on work, you can use the Sublime Merge GUI, which gives you a graphical representation and visual history of the commits in your repo.

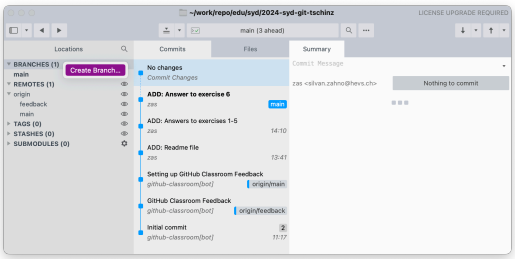
1. Create a development branch `dev01` in your local repo.

Commandline

```
git checkout -b dev01
```

GUI

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



2. Create a commit on this branch:
   - To create and populate a `hello_world.py` file.

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

```
git add hello_world.py
git commit -m "ADD: hello_world.py"
```

3. Checkout of the `main` branch

```
git checkout main
```

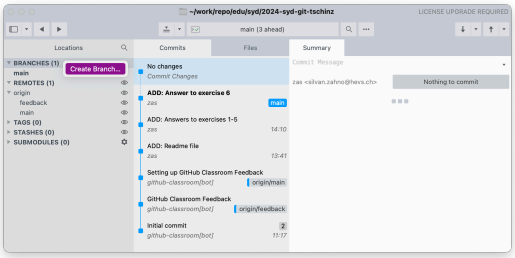
4. Starting from the main branch, create a new development branch `dev02`.

Commandline

```
git checkout -b dev02
```

GUI

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



5. Create a commit on this branch:
   - To create and populate a `hello_world.rs` file.

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

```
git add hello_world.rs
git commit -m "ADD: hello_world.rs"
```

6. Checkout of the `main` branch.

```
git checkout main
```

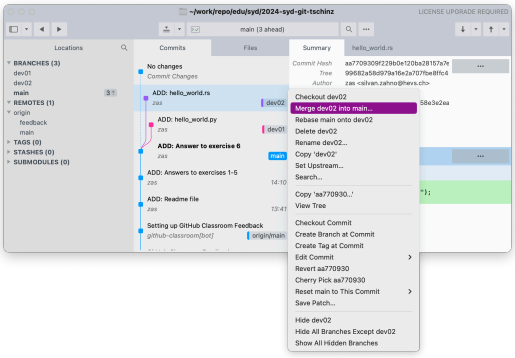
7. Merge the `dev02` branch into `main`.

Commandline

```
git merge dev02
```

GUI

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



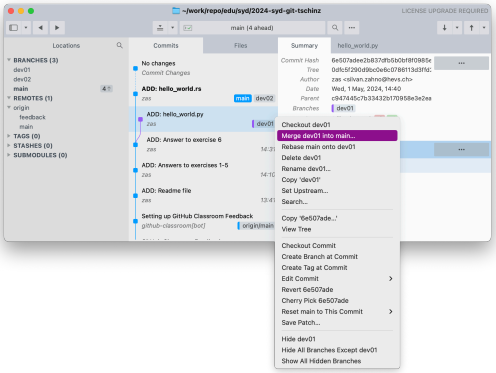
8. Merge the branch `dev01` into `main`.

Commandline

```
git merge dev01
```

GUI

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



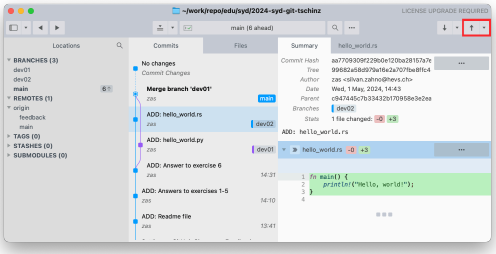
9. Push your local repository to your cloud GitHub repository.

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`



8.1.1 End result

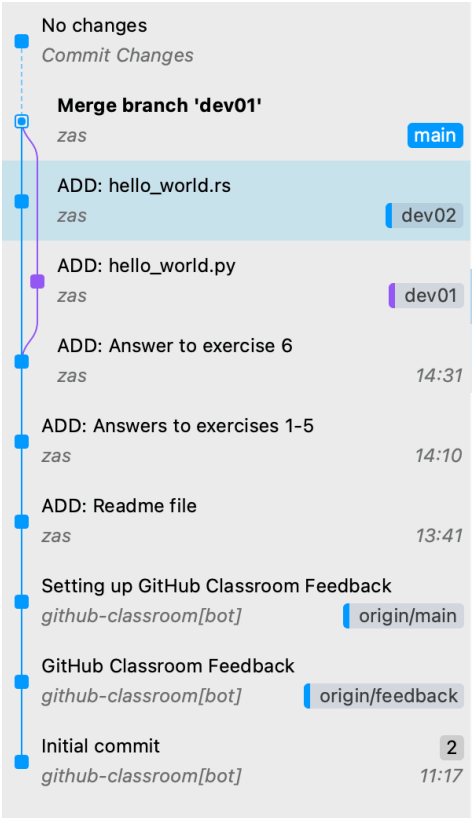


Figure 18: Status after repo branches `dev01` and `dev02` have been merged

The status of your repository should be similar to Figure 18. This is part of the evaluation.

9 Gitgraph

The Figure 19 shows an example of a git repository. Name all the elements that can be seen in the image (points 1-10).

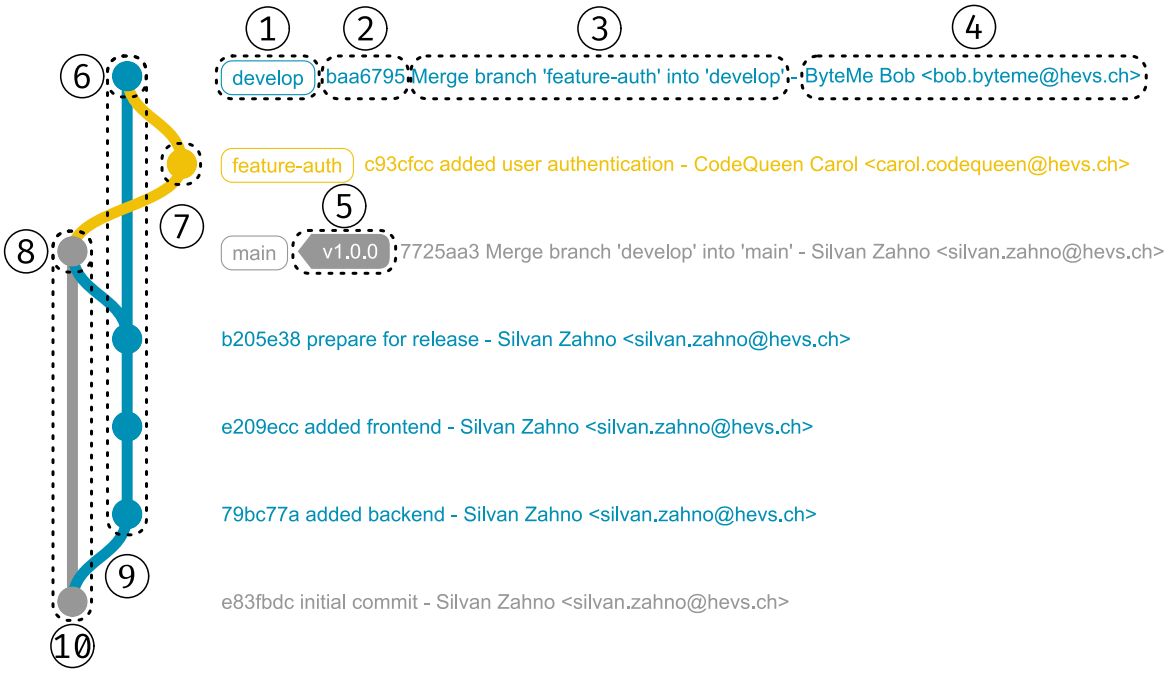


Figure 19: Example of a Gitgraph

Task 7

1. Name all elements that can be seen in the image Figure 19 (points 1-10).
2. Push your local repository to your cloud GitHub repository.

10 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/syd-gitflow [4]
This is a public Git repo hosted on Github.

10.1 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.

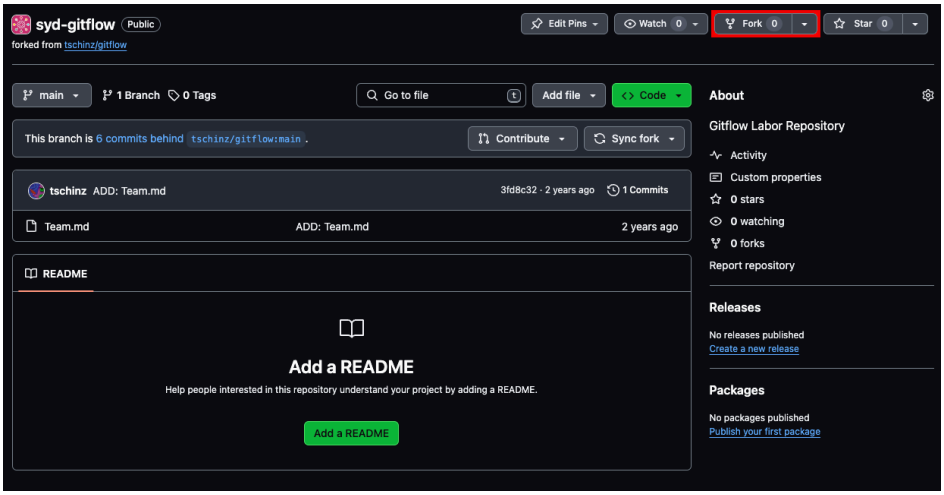


Figure 20: Fork button for a GitHub repository

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

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

10.2 Parallel collaboration

In a local feature branch, edit the `Team.md` file. Replace your given number with your first name and last name.

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

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

10.3 Pull Request

Create a “pull request” (merge request) on GitHub. Use the interface on the GitHub website to do this.
Once all pull requests are ready, merges will be done in consultation with the entire group (and teachers).

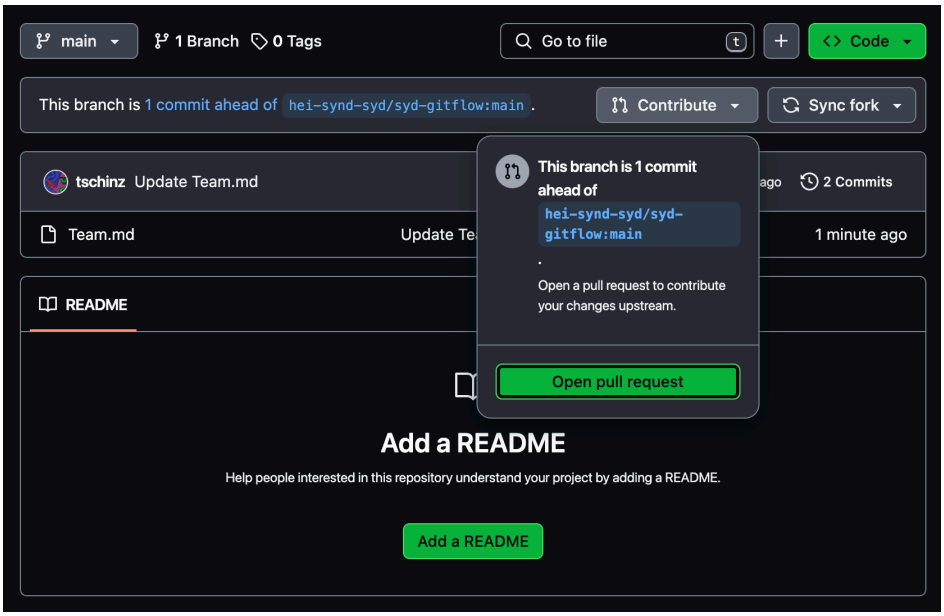


Figure 21: Create Pull Request on Github

11 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

11.1 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.

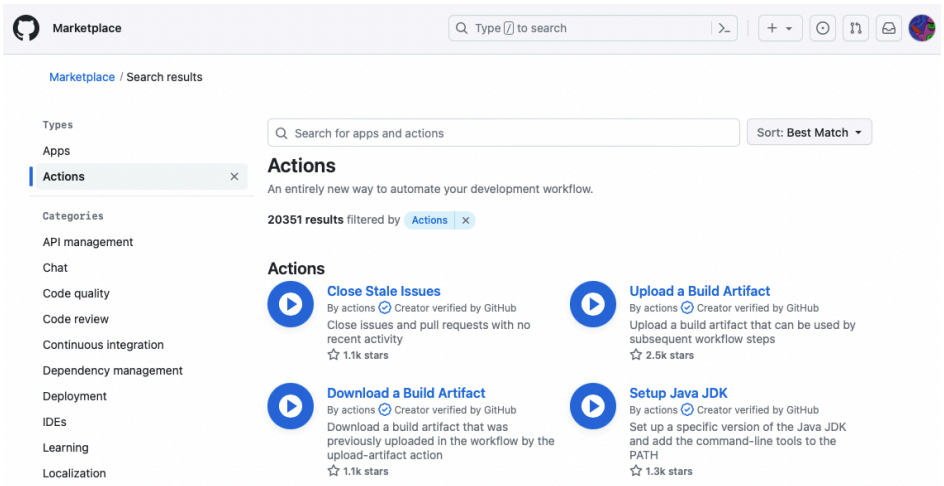


Figure 22: Github Action Marketplace

11.2 Learn Git Branching

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

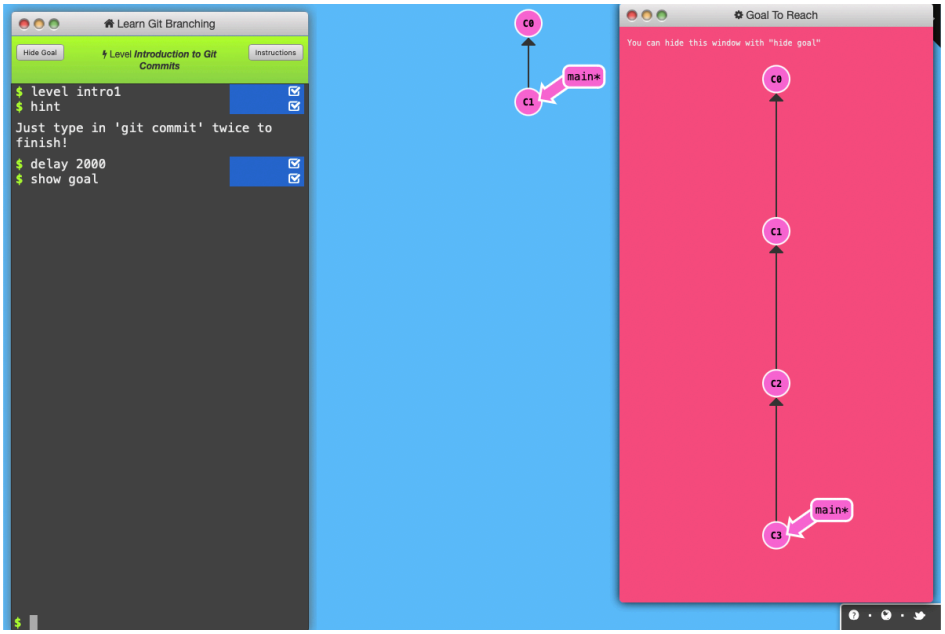


Figure 23: Learn git branching website

A GIT commands

Github git cheatsheet [5], [6]

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

Bibliography

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

[2] "GitLab Hevs." Accessed: Apr. 25, 2023. [Online]. Available: https://gitlab.hevs.ch/

[3] "Sublime Merge - Git Client from the Makers of Sublime Text." Accessed: Apr. 25, 2023. [Online]. Available: https://www.sublimemerge.com/

[4] tschinz, "Tschinz/Gitflow." Accessed: Apr. 25, 2023. [Online]. Available: https://github.com/tschinz/gitflow

[5] gitlab, "Git Cheatsheet." 2023.

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