Introduction to git - Part A & B

Installation & Setup

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], in particular the tool Sublime Merge and optionally the Git command line tool, which have to be installed and configured on your computer (see Section 2). In addition, an account is created on the Github platform.
Finally, we will learn the basics of Markdown in Section 6.6 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 and/or Sublimemerge. You can choose whether you want to use the command line or the GUI during the lab.

2.1 Sublime Merge

Visit the website https://www.sublimemerge.com and download and install the Sublime Merge tool [2].

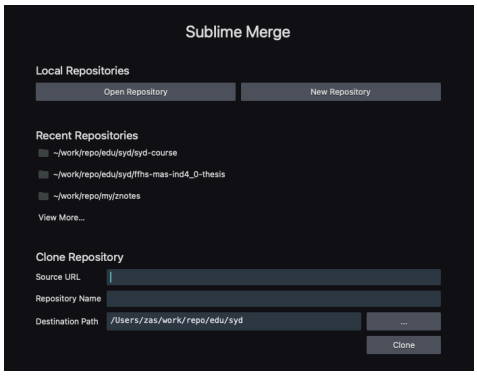


Figure 1 - Sublime Merge GUI

2.1.1 Configuration

When cloning a repository, Sublime Merge will automatically ask for your identity and prompt you to log in to your Github account.

2.1.2 Overview

The interface of Sublime Merge is presented in the Figure 2:

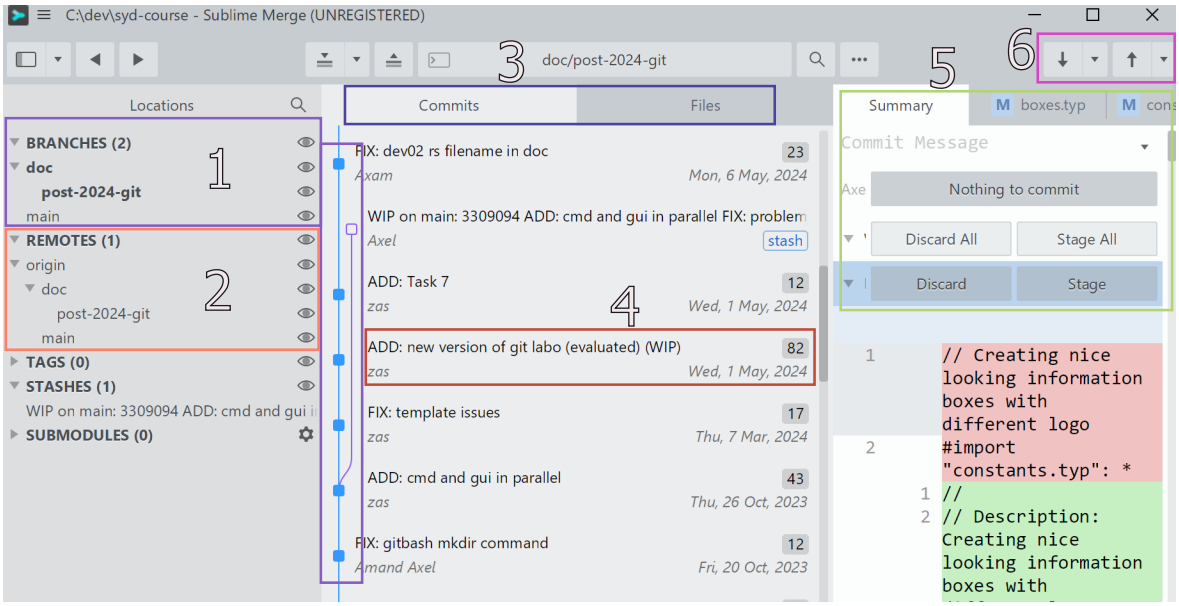


Figure 2 - Sublime Merge GUI

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.

2.2 `git` - command line

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.

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.

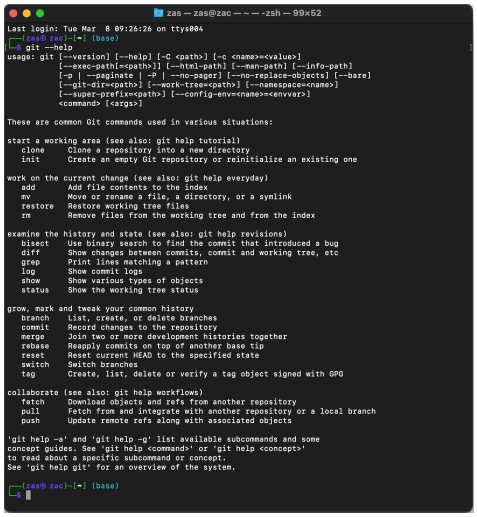


Figure 3 - git Terminal

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

```
git --help
```

2.2.1 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
```

2.3 Online accounts

2.3.1 Github

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

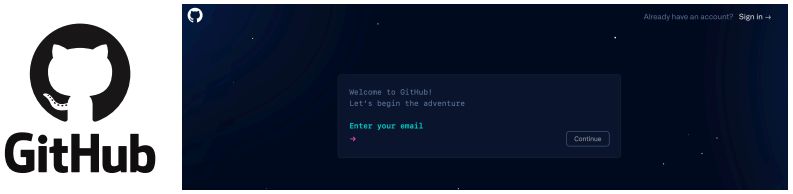


Figure 4 - GitHub Login

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**"

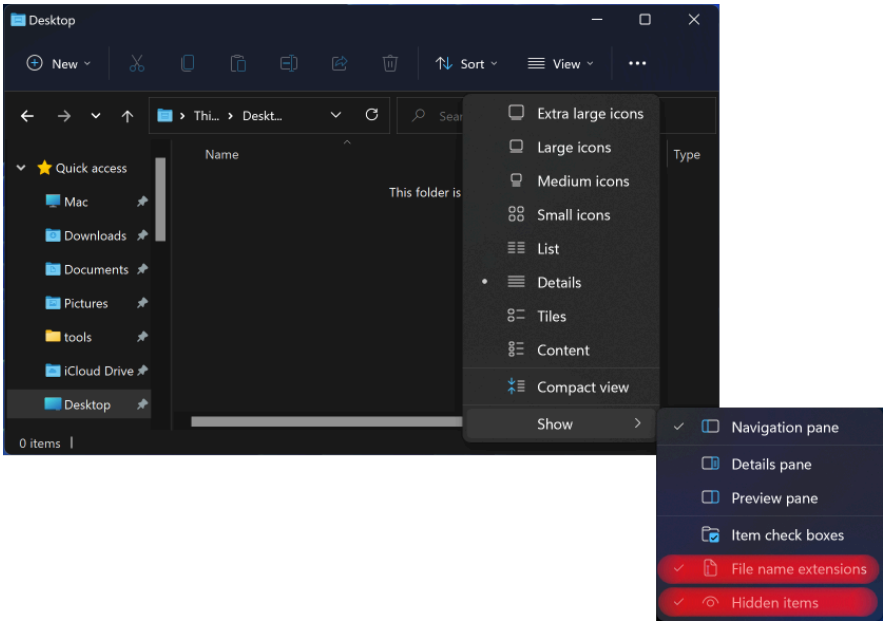


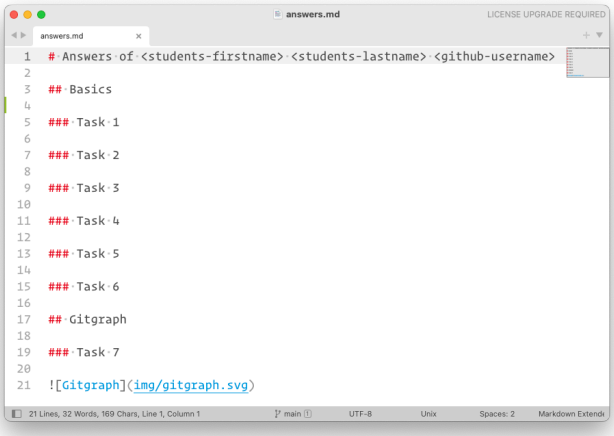
Figure 5 - Windows File Explorer Configuration

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.



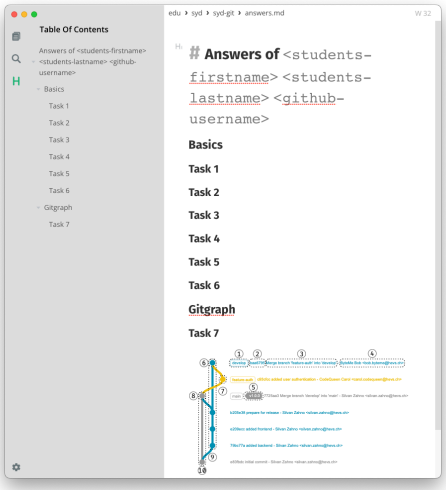


Table 1 - Left: Simple Texteditor (Sublime Text), Right: Marktext

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 https://wiki.zahno.dev/multimedia/writing/md/md_github.html.

```
# 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

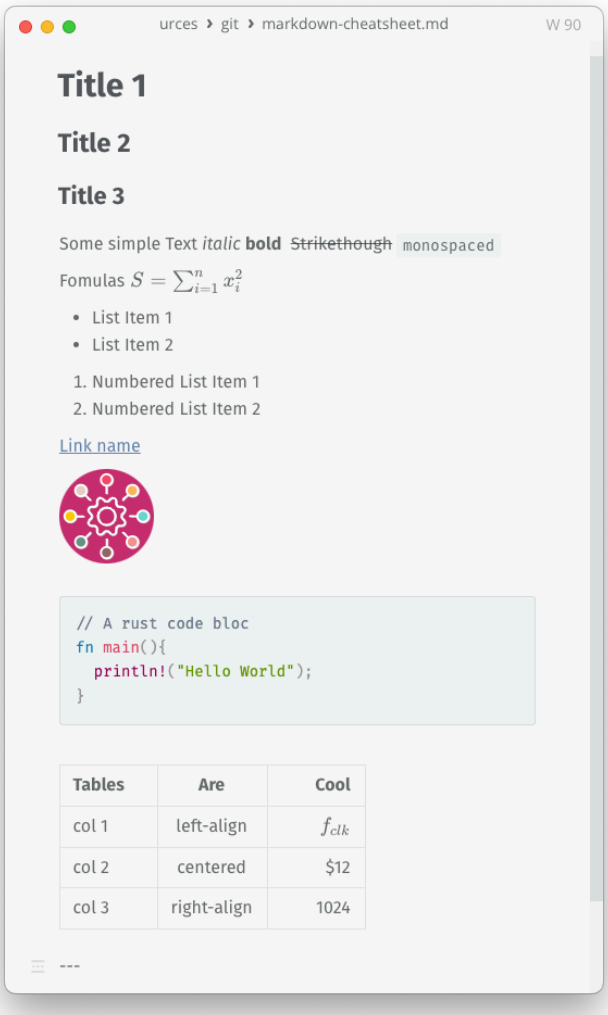
[Link name](https://hevs.ch/synd)

![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. You should have:

- [ ] Installed Git and Sublime Merge
- [ ] Configured git with your name and email
- [ ] Created a GitHub account
- [ ] Familiarized yourself with the Sublime Merge graphical user interface (GUI)
- [ ] Familiarized yourself with the Git theory and commands
- [ ] Familiarized yourself with Markdown and its syntax

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 6 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 7. In Section 9 we all work together on a repository. Finally, there is some optional work in Section 10.

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.

5.1 Tools

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

Figure 6 - Git CMD Commandline

Figure 7 - 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 ...`**

5.1.1 Overview

The interface of Sublime Merge is presented in the Figure 8:

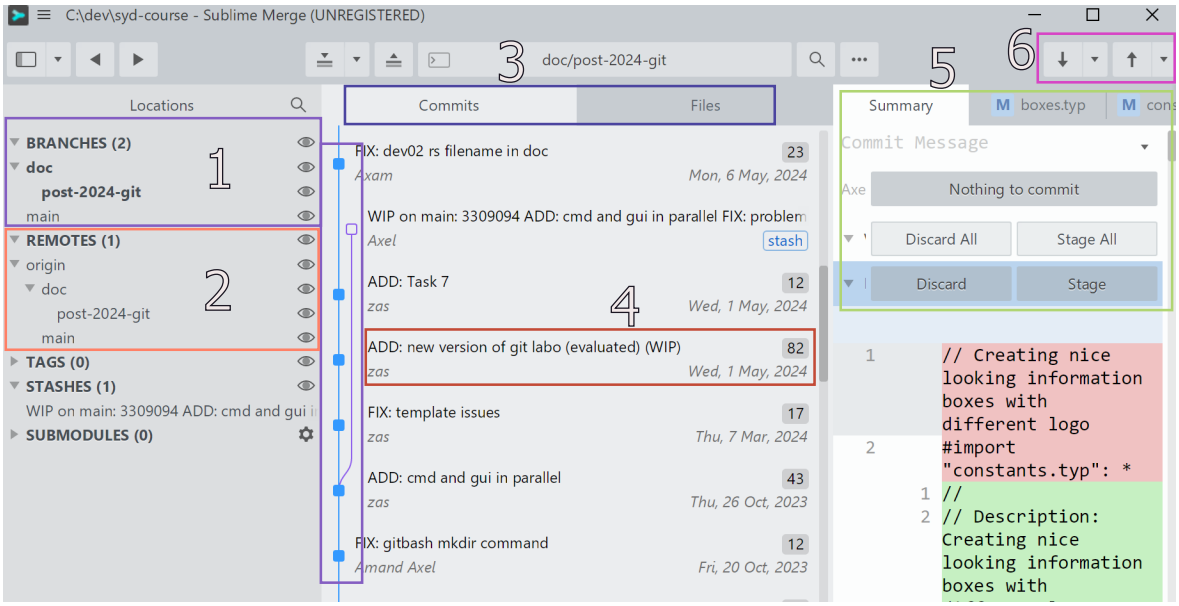


Figure 8 - Sublime Merge GUI

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.

6 Basic Operations

6.1 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 9* . The changes made are reflected in your own repository, not in the one that was copied. So you can work without fear of creating synchronization problems.

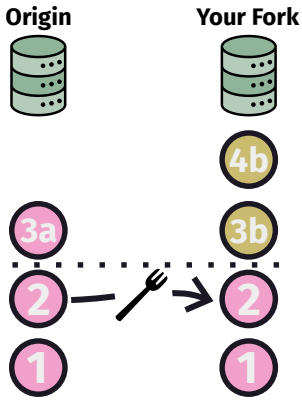
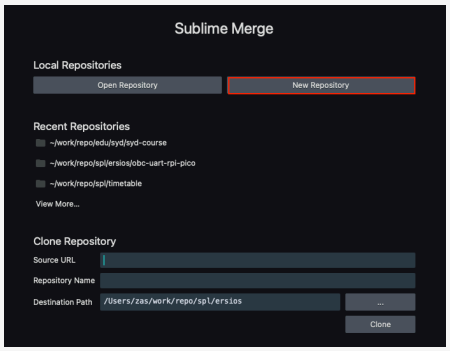


Figure 9 - Fork of the template repository

If you want to create a new repository from scratch (not needed in this labo), you can do so with the following command 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.

Create a fork of the template repository using the following link https://classroom.github.com/a/rYLfVeC0.

For this, you will need to log in to GitHub. Accept the Classroom invitation - *Figure 11* , which will give you the link to YOUR own repository - *Figure 12* .

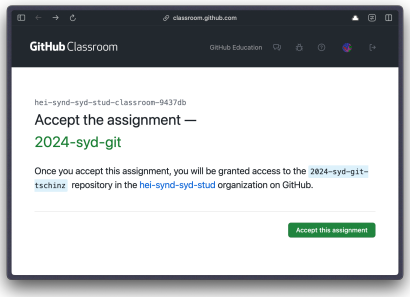


Figure 11 - Invitation link for forking

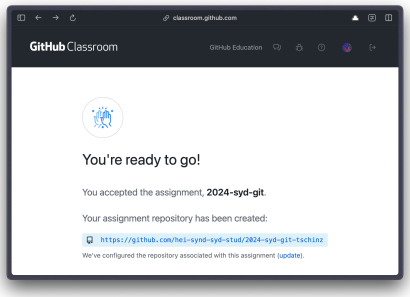


Figure 12 - Link to clone the repository

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.

6.2 Clone

After you have created the fork, you will receive a link to your own repository - *Figure 12* . Clone it to your local computer at a location of your choice.

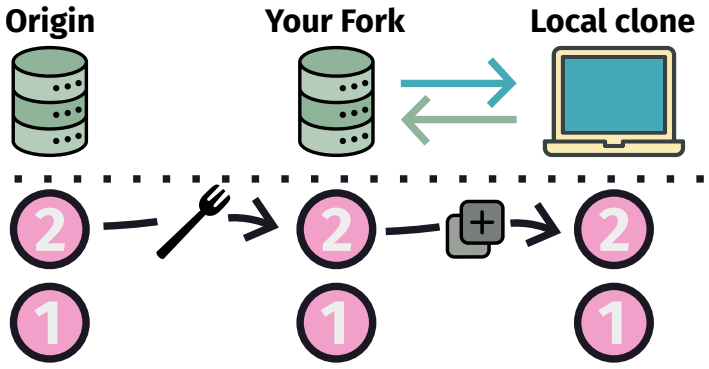


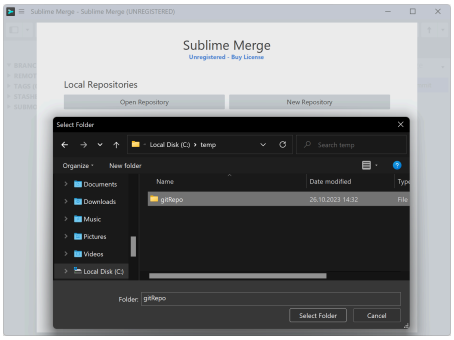
Figure 13 - Cloning

Commandline

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

# Example
git clone https://github.com/hei-synd-
syd-stud/2024-syd-git-tschinz.git
```

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



6.3 Get status

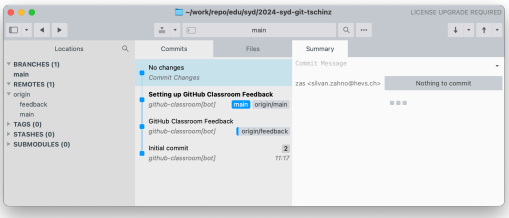
Get information about your repo with the following commands:

Commandline

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

GUI

See the status in the main window.



Currently, your repository has no changes since nothing has been modified. In Sublime Merge, at the top of the window, it says **`No changes`**.

6.4 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

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

Like shown under Section 6.3, check the status of your repo.

Task 1

What is the status? What does it mean?

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

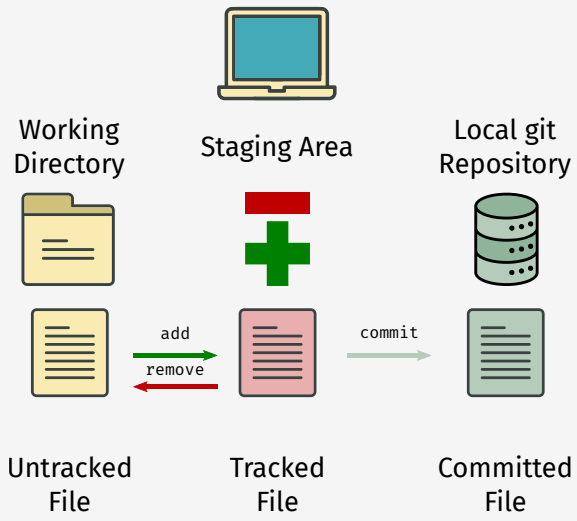


Figure 14 - Types of local Git operations

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.

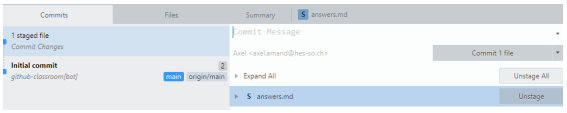
To save the changes *locally*, you first need to select the changes that will be savedin the repo. This process is called **`Stage`**.

Commandline

```
git add answers.md
```

GUI

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



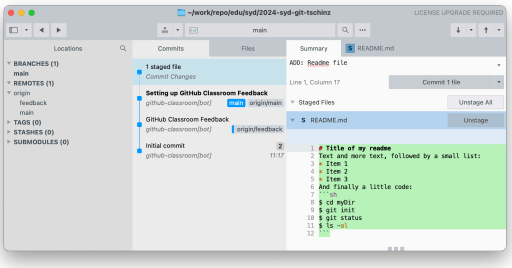
Once all changes are selected, the save is done through a **`Commit`**. A commit is a snapshot of your repo at a given time and once done, the state of the folder is saved in the repository. The **`Commit`** is identified by a unique identifier (hash) and contains information about the author, date and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



Task 2

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

- What does the example commit message mean?
- Is it possible to create a commit without a message?
- What changes in the repo now?
- Is the remote repo (here Github) up to date with our changes?

6.5 Add files

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

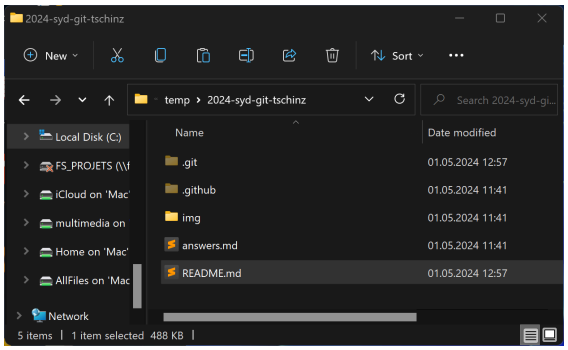
Commandline

```
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:

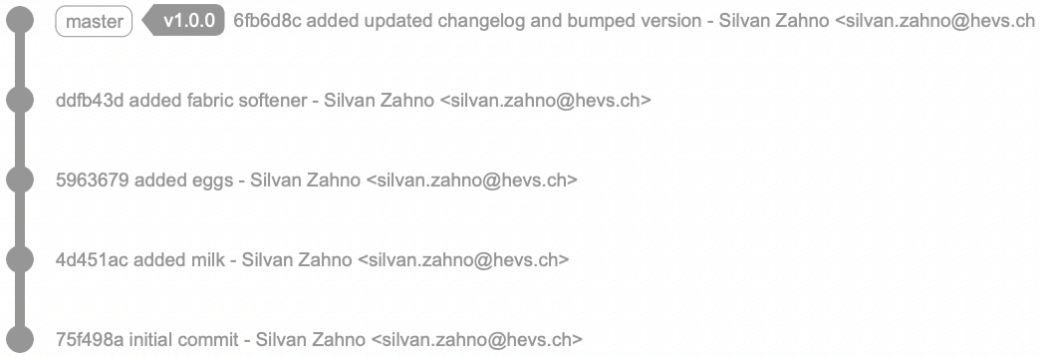
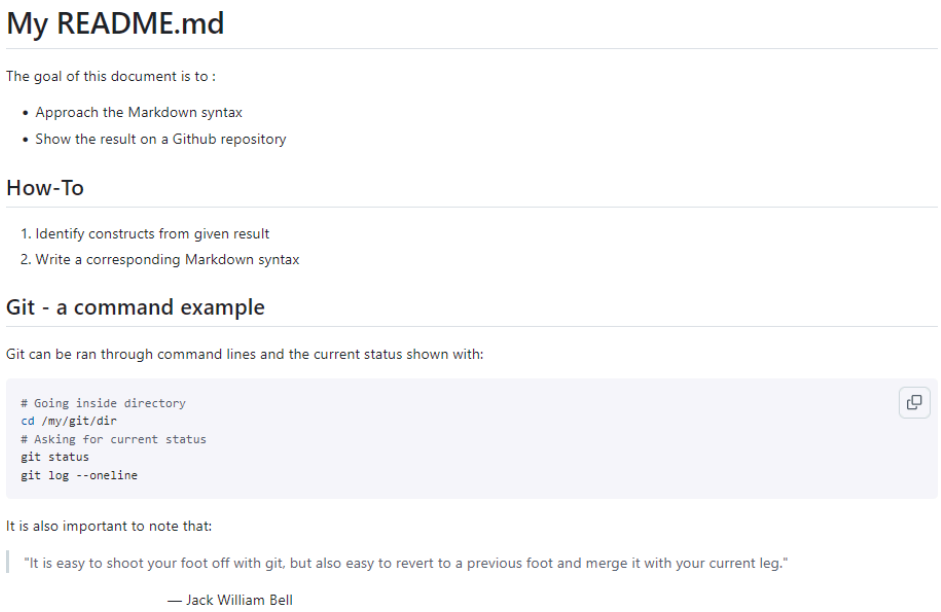


Figure 15 - Five commits on the local repo, each commit has its own identification

6.6 Markdown

Modify the **`README.md`** file with a text editor to make it resemble roughly like t o 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 16 - Markdown structure to reproduce

To help you, you can use sites like https://markdownlivepreview.com/ to preview the result.

Here is a Markdown cheatsheet 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.

6.7 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:

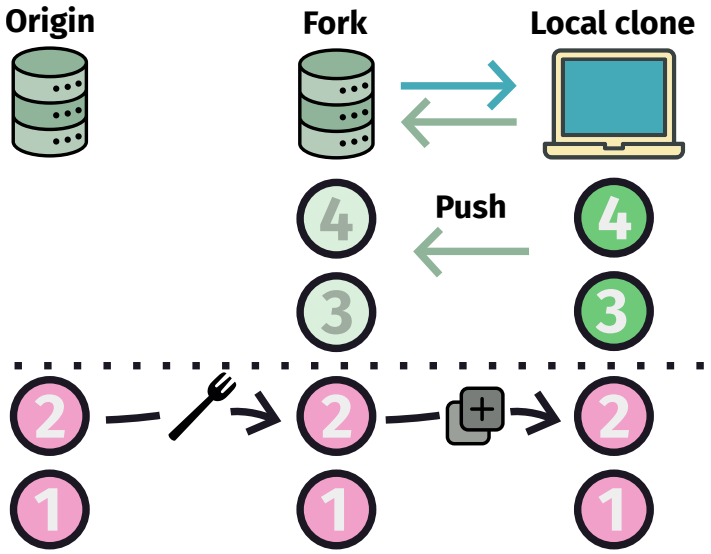


Figure 17 - Push to the remote repository

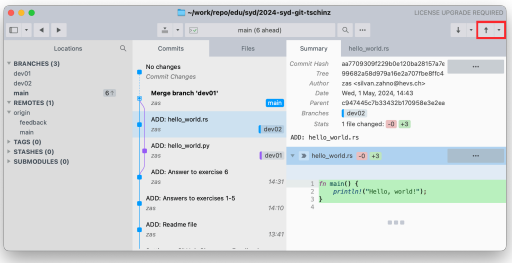
To do this, once all changes have been committed:

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.

6.8 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
```

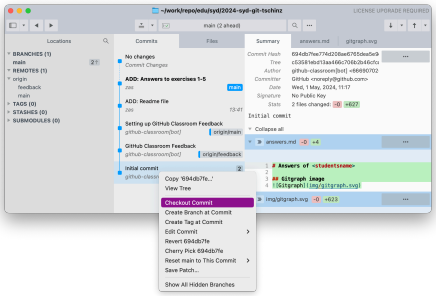
To change commits, perform the **`checkout`** operation:

Commandline

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

GUI

`Select First Commit (Initial Commit)` ⇨ **`Right click`** ⇨ **`Checkout commit`**



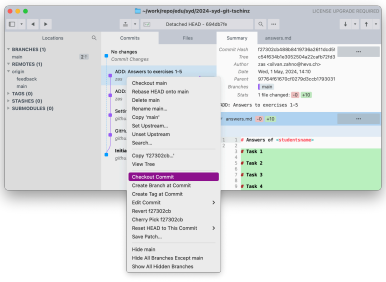
Then go back to the last commit in the same way. Apart from the hash, it is possible to return to the last commit by using the name of the branch, here **`main`**.

Commandline

```
git checkout main
```

GUI

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



Task 4
What happens when you go back to the "Initial commit" commit? What happens when you go back to the last commit?

6.9 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?

6.10 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

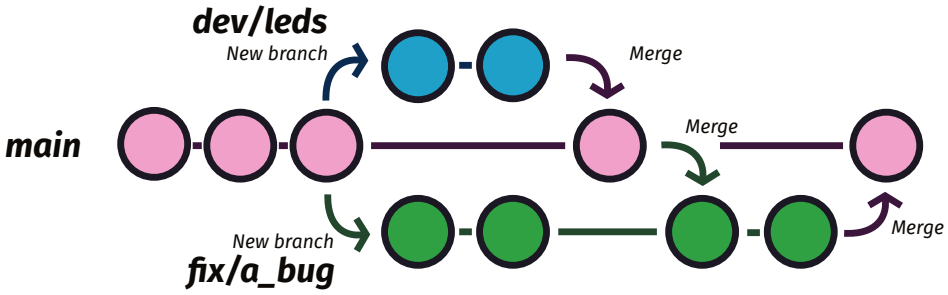
```
git tag chapter2
git push origin chapter2
```

GUI

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

7 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.

7.1 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 ...

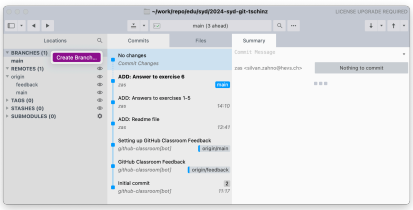
1. Create a development branch **`dev01`** in your local repo:

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:

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

   - Make a commit as presented in chapter Section 6.4, with a clear message.

7.2 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.

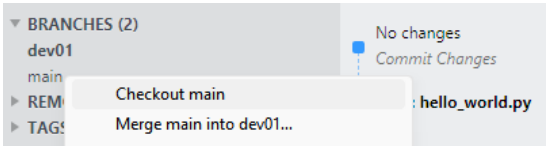
1. Switch back to the **`main`** branch:

Commandline

```
git checkout main
```

GUI

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



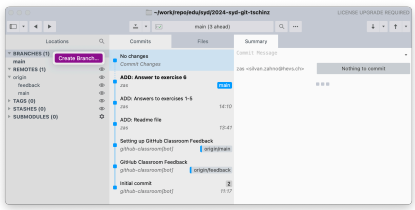
2. From the main branch, create a new development branch **`dev02`**:

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 6.4, with a clear message.

7.3 Merge dev02

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

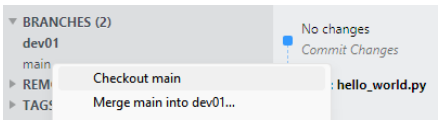
1. Switch back to the **`main`** branch:

Commandline

```
git checkout main
```

GUI

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



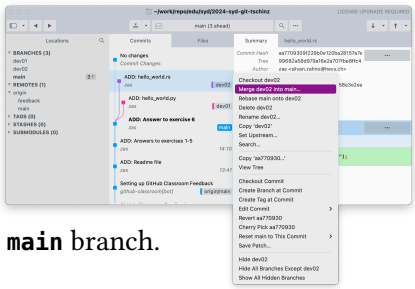
2. Merge the **`dev02`** branch into **`main`**:

Commandline

```
git merge dev02
```

GUI

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



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

7.4 Merge dev01

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

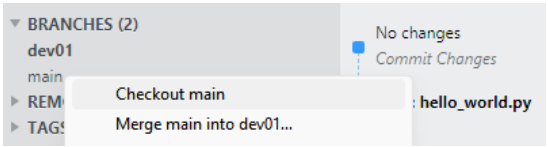
1. Ensure you are on the **`main`** branch:

Commandline

```
git checkout main
```

GUI

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



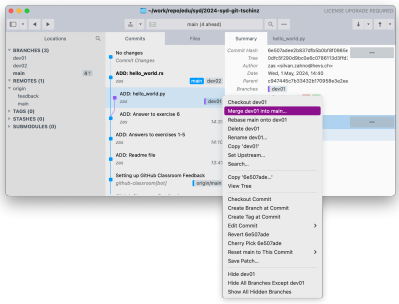
2. Merge the **`dev01`** branch into **`main`**:

Commandline

```
git merge dev01
```

GUI

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



7.5 Final Result

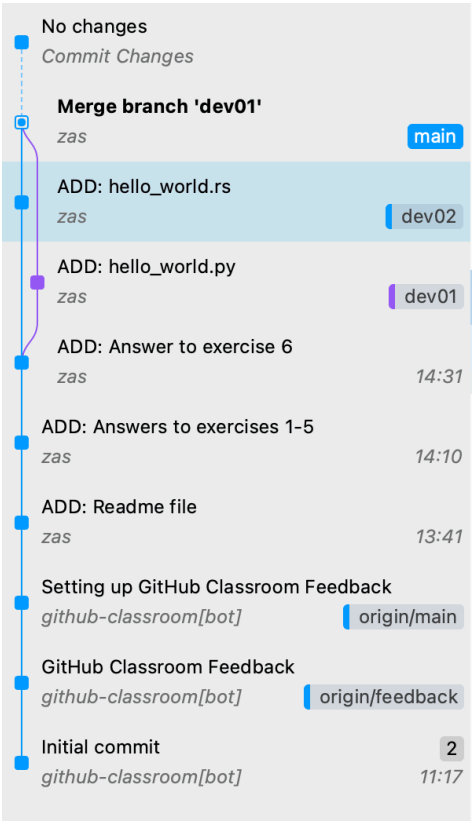


Figure 27 - Status after repo branches **`dev01`** and **`dev02`** have been merged

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.

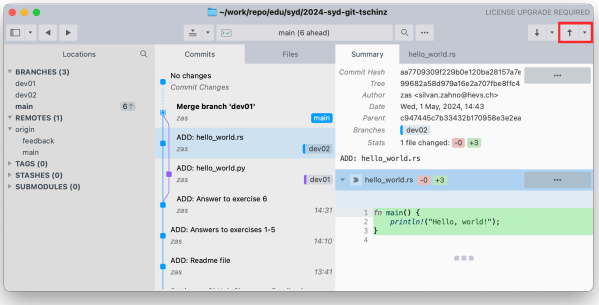
1. Push your repository to the remote GitHub server. **Three `push` commands are required to push each branch separately**:

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 6.10. Don't forget to push the tag!

The status of your repository should be similar to Figure 27 in addtions the tags **`chapter2`** and **`chapter3`**. This is part of the evaluation.

8 Gitgraph

The Figure 29 shows an example of a git repository. Name all the elements that can be seen in the image (points 1-10).

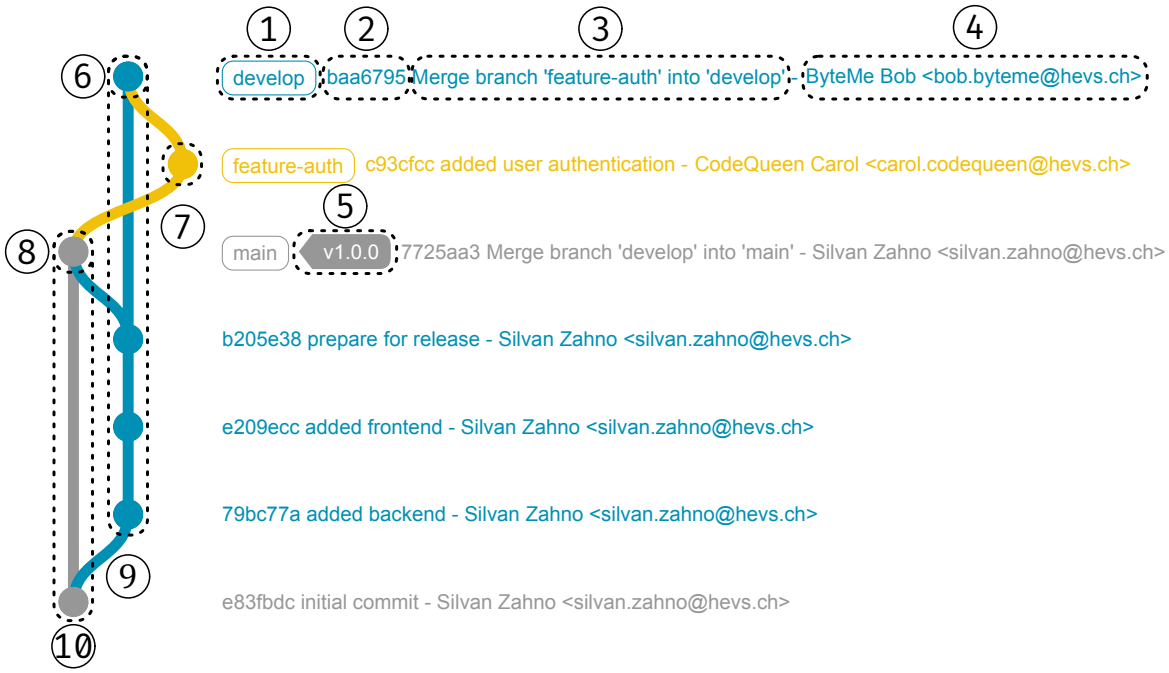


Figure 29 - Example of a Gitgraph

Task 7

1. Name all elements that can be seen in the image Figure 29 (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 6.10

Congratulations, the work on your first GitHub repository is finished.

9 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 [3]

This is a public Git repo hosted on Github.

9.1 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.

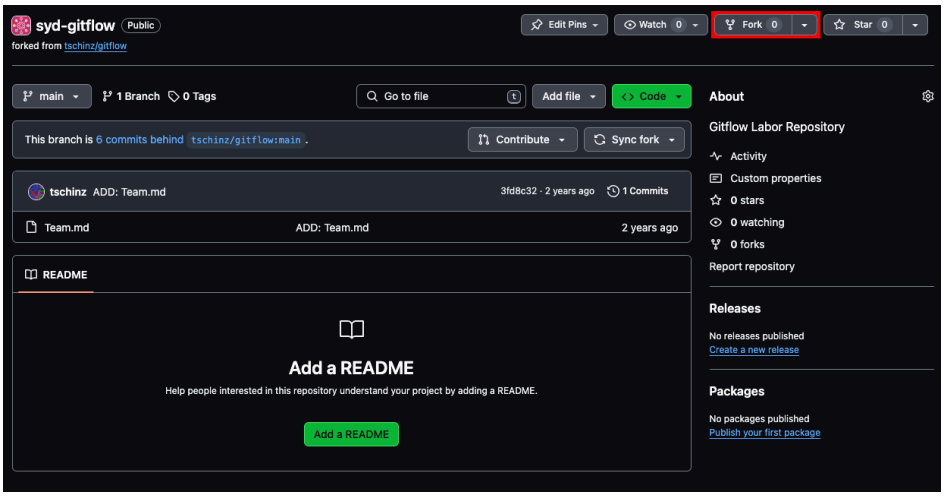


Figure 30 - Fork button for a GitHub repository

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

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

9.2 Parallel collaboration

In a local feature branch, edit the **`Team.md`** file. Replace your given number with your first name and last name. **`Commit`** and **`push`** your branch to your fork of the repository on GitHub.

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!

9.3 Pull Request

Now that your changes are ready, it is still necessary to perform a merge on the original repository. As you do not have the rights, it is possible to make a merge request - **`pull request`** - on the original repository.
This way, the administrator can accept the changes, ask you to correct items before accepting the merge, discuss issues, etc.
For this, use the interface of the GitHub website. **`Contribute`** ⇨ **`Open pull request`**.

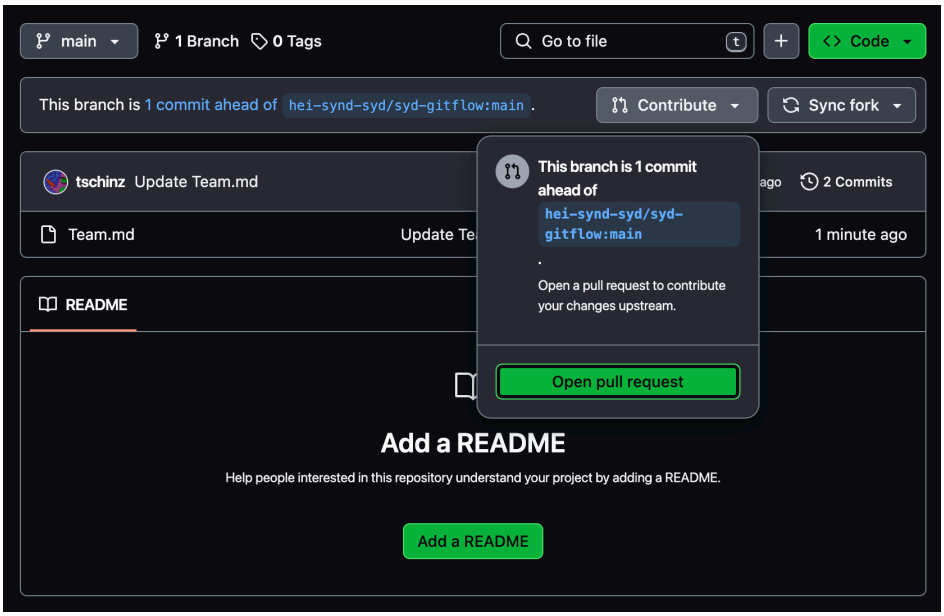


Figure 31 - Create Pull Request on Github

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 3.

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 assistans.

10 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

10.1 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.

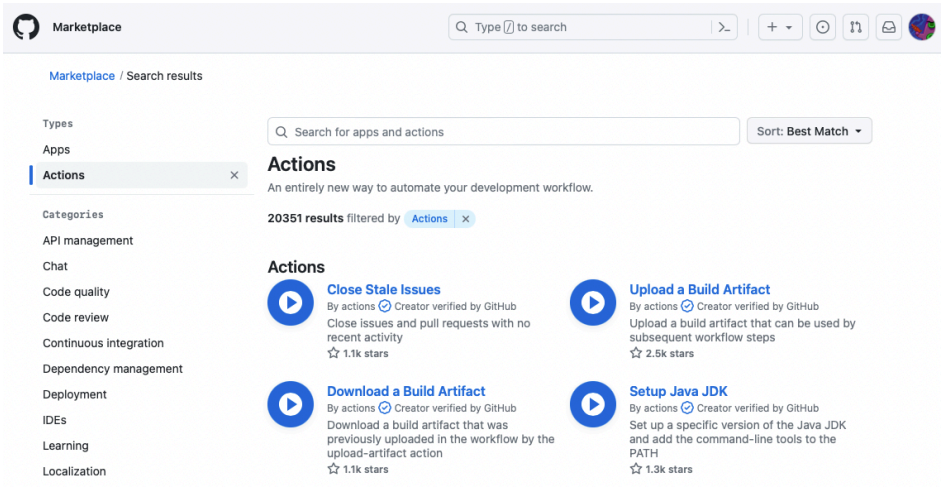


Figure 32 - Github Action Marketplace

10.2 Learn Git Branching

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

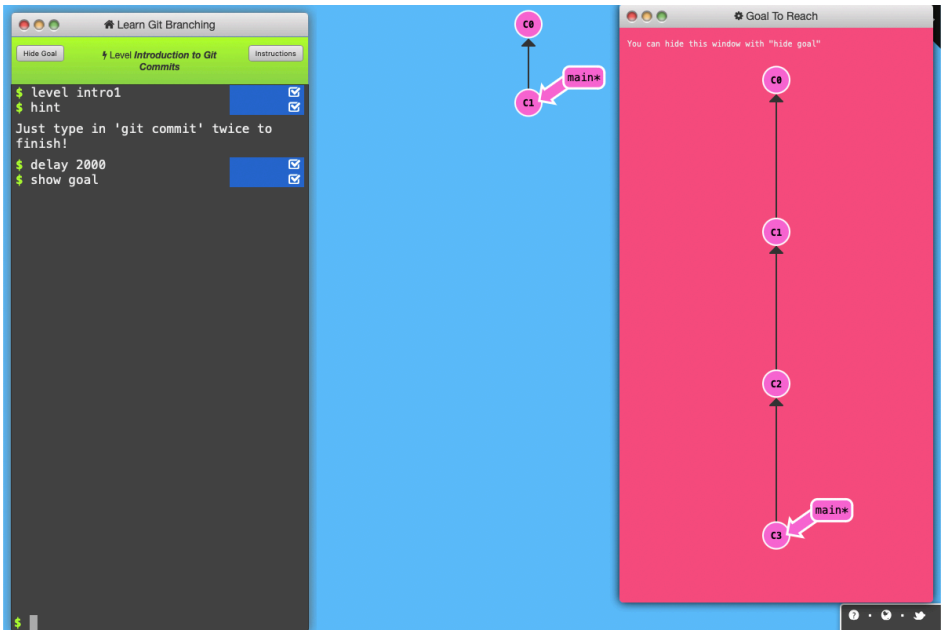


Figure 33 - Learn git branching website

Bibliography

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

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

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

[4] gitlab, “Git Cheatsheet.” 2023.

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

11 Appendix

A GIT commands

Github git cheatsheet [4], [5]

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