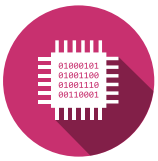
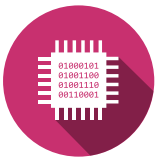



# Introduction to git file versioning


# Contents




# 1 Goal

In this lab we will learn the basic principles of version control `git` [1].

To begin, we will install and configure Git on your machine see Section 2. As well as create accounts on Github [2] and Hevs Gitlab [3].

In Section 3 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` as well as `merge` will be tried in an example in Section 4.

In Section 5, we all work together using the gitflow principle.

Finally, there are some optional works in Section 6.

The answers to the questions should be written down in a Markdown file. The file should be named **`answers.md`**.

# 2 Installation

The first step is to install Git as well as Sublimemerge.

## 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." This is a Unix/Linux-like command editor that allows you to run Git commands in console mode. This is the interface we will use in this lab.

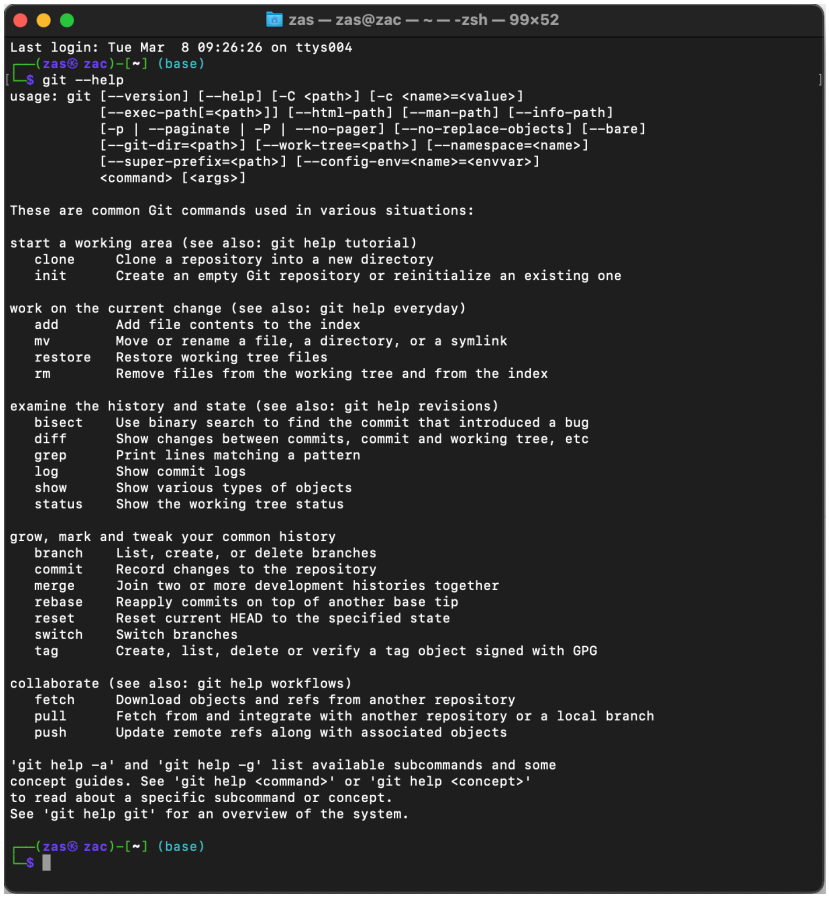



Figure 1 - git Terminal

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
```

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 [4].

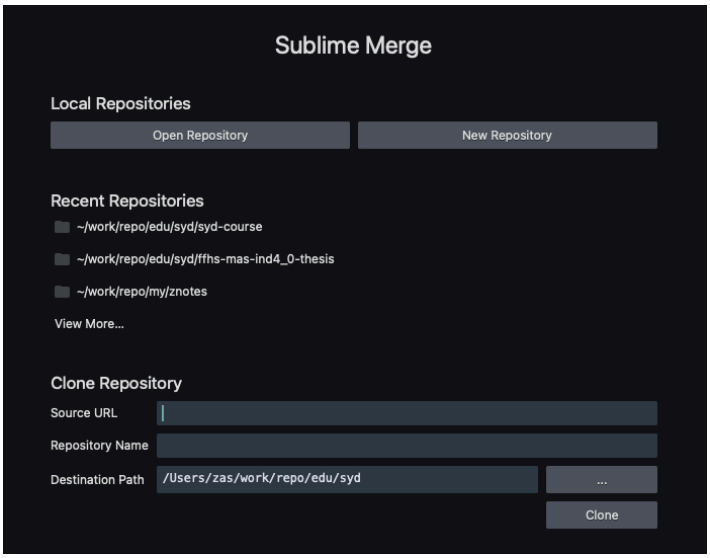



Figure 2 - Sublime Merge GUI

## 2.3 Online accounts

### 2.3.1 Gitlab

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

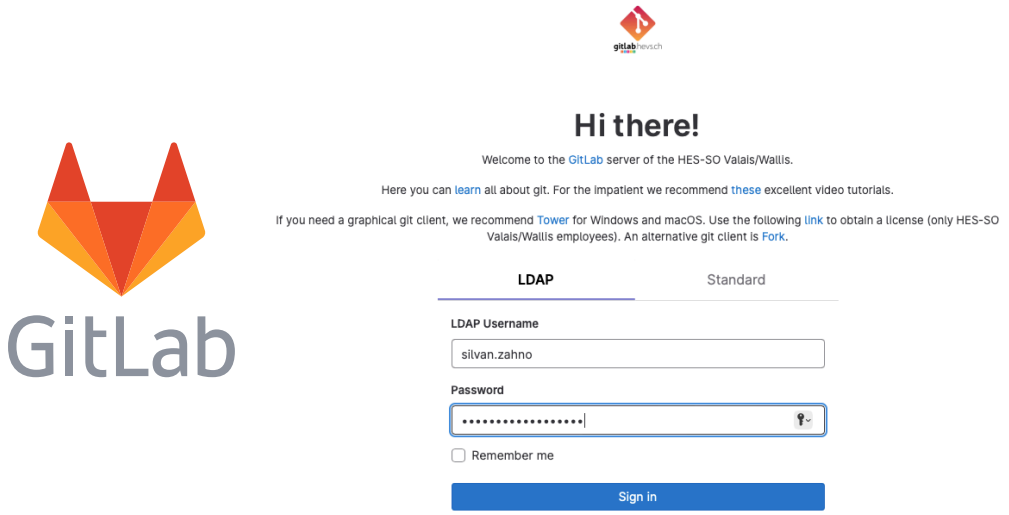



Table 1 - Gitlab Login

### 2.3.2 Github

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

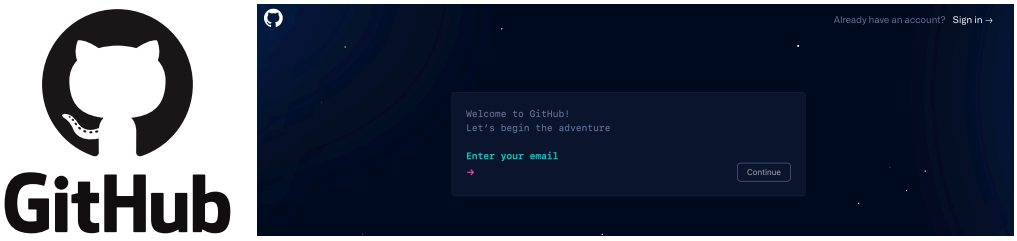



Table 2 - GitHub Login

## 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 3:

File Explorer ⇨ View ⇨ Show ⇨ Activate "**File name extensions**" and "**Hidden items**"

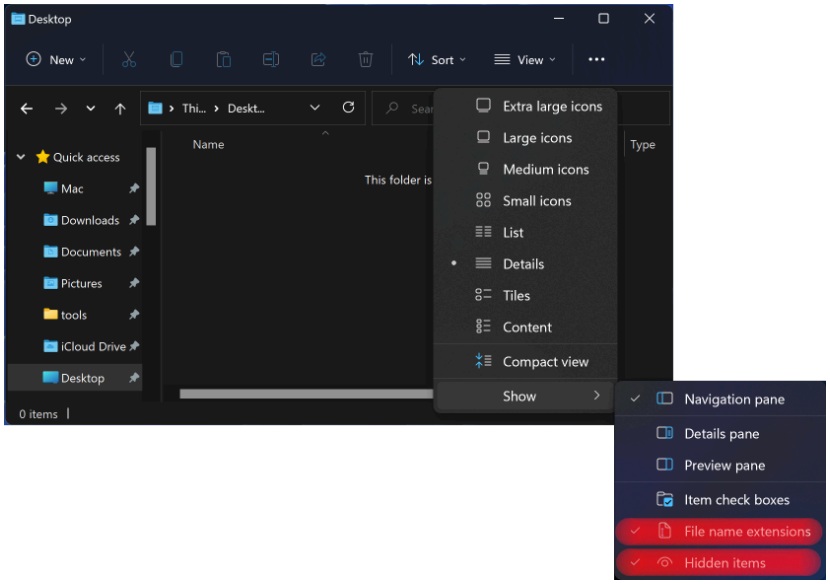



Figure 3 - Windows File Explorer Configuration


# 3 Basic operations

## 3.1 Create a git repository.

Using the Git Bash console, create an empty directory on your computer, for example `C:\temp\gitRepo` or `~/tmp/gitRepo`. This directory will be your git repository.

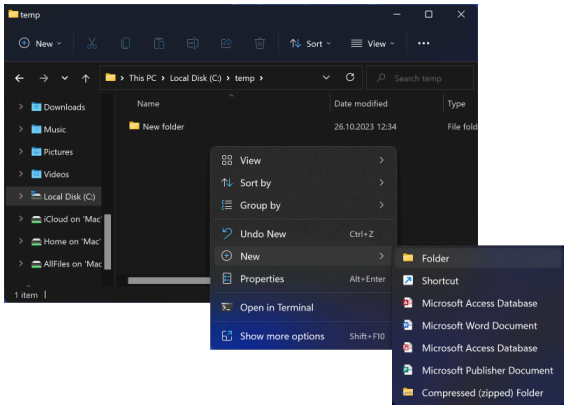
You can use the Unix/Linux commands `ls` (list files and directories), `cd` (change directory), `pwd` (print current working directory), and `mkdir` (make directory) to create this directory.

**Commandline**

```
mkdir -p c:/temp/gitRepo
cd c:/temp/gitRepo
```

**GUI**

**`C:\temp` ⇨ `New` ⇨ `Folder` ⇨ `gitRepo`**




## 3.2 Initialize

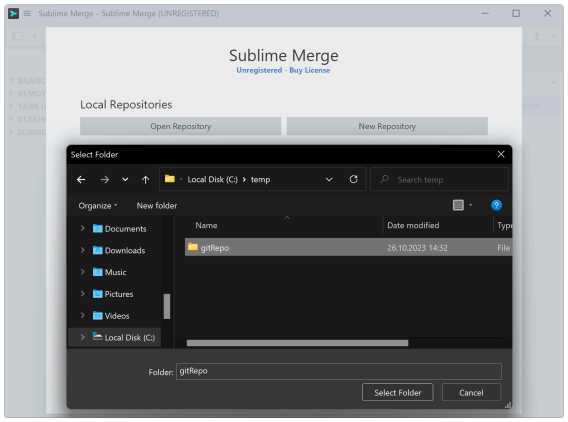
Once you are in that directory, initialize it as a git repo with the command:

**Commandline**

```
git init
```

**GUI**

**`CTRL+T` ⇨ `New Repository` ⇨ `Select Folder`**




What changed in the directory after you initialized it as a git repo?

Write down the answer!

## 3.3 Get status

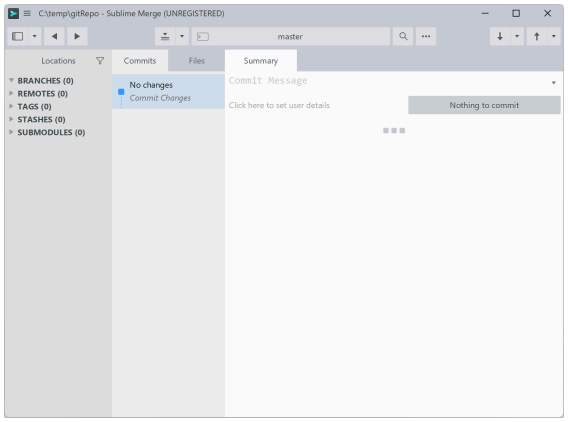
Get information about your repo with the following commands:

**Commandline**

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

**GUI**

See the status in the main window.




## 3.4 Add file

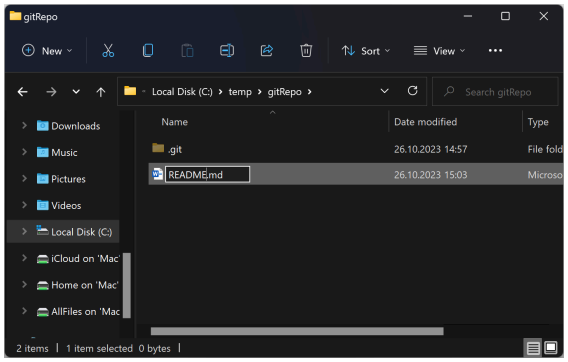
Now create an empty file named **`README.md`** in the root directory of your repo.

**Commandline**

```
touch README.md
```

**GUI**

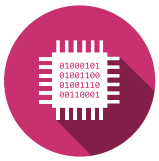
**`C:\temp`** ⇨ **`New`** ⇨ **`Any File`** ⇨ **`README.md`**




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

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.

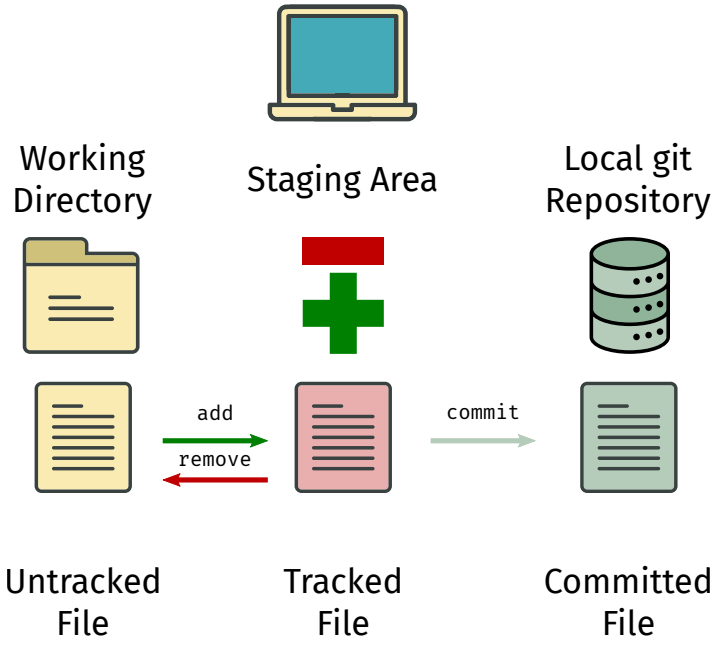



Figure 4 - Types of local Git operations

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.

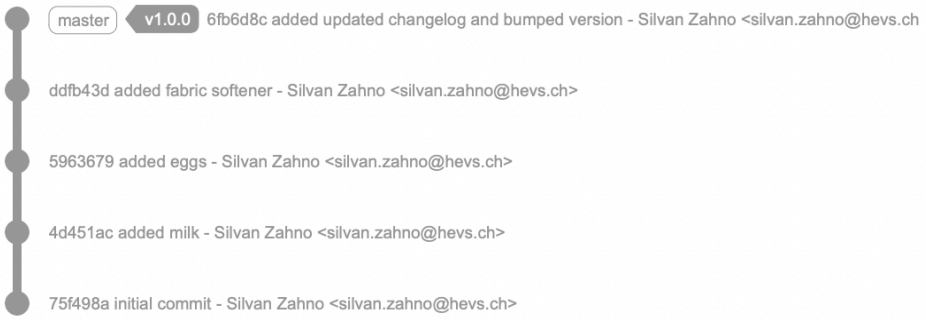



Figure 5 - Five commits on the local repo, each commit has its own identification

## 3.5 Add file to repo

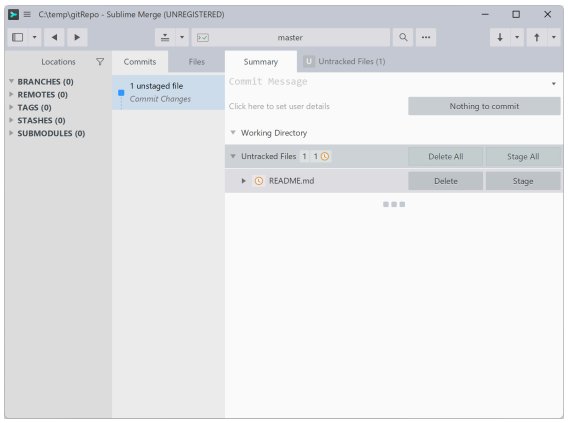
Add the previously created file **README.md** to the stage with the command:

**Commandline**

```
git add README.md
```

**GUI**

**Untracked Files** ⇨ **README.md** ⇨ **Stage**




Retrieve the information about your repo again, what do you find?

> 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
```
```

Get the information about your repo again, what do you find?

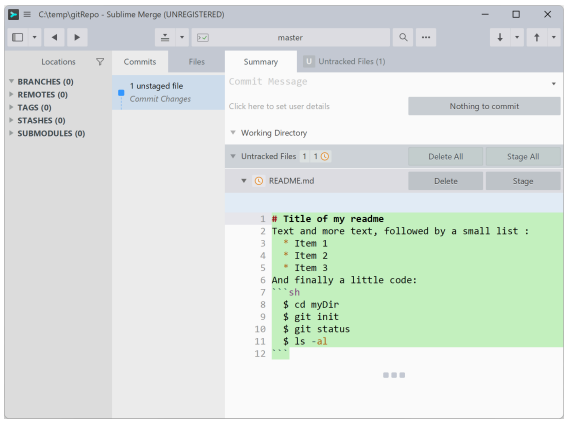
> Write down the answer!

## 3.6 Add new changes

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

| Commandline | GUI |
| --- | --- |
| | **`Untracked Files`** ⇨ **`README.md`** ⇨ **`Stage`** |

```
git add README.md
```




## 3.7 Execute commit

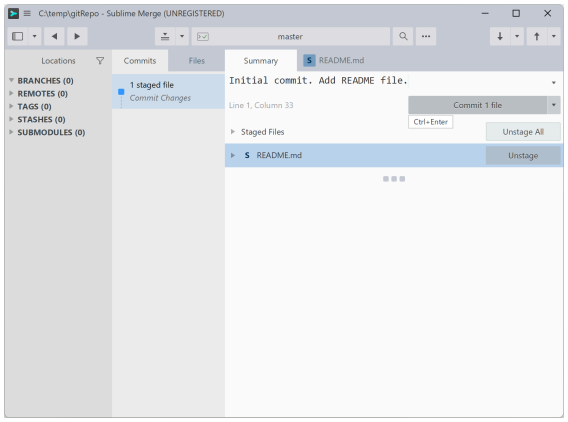
Now perform a commit with the following command:

**Commandline**

```
git commit -m "Initial commit. Add README file."
```

**GUI**

**`Commit Message` ⇨ `Initial commit. 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.

## 3.8 More information

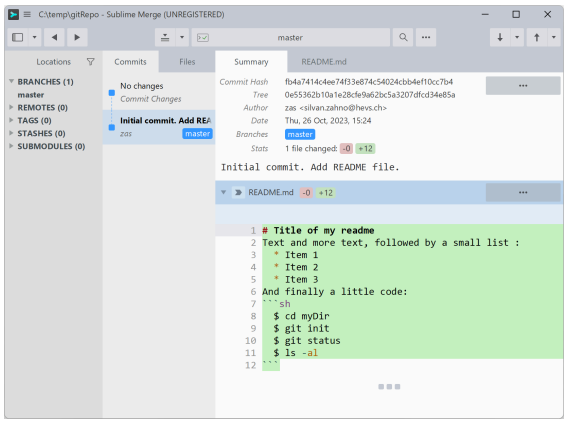
What information can you get now with the command:

**Commandline**

**GUI**

**`Select First Commit` ⇨ `See all informations`**

```
git log --oneline
```




Clearly explain all the information in this line.

Write down the answer!

## 3.9 More commits

Perform another commit to add a new (empty) **`hello_world.py`** file to your repo.

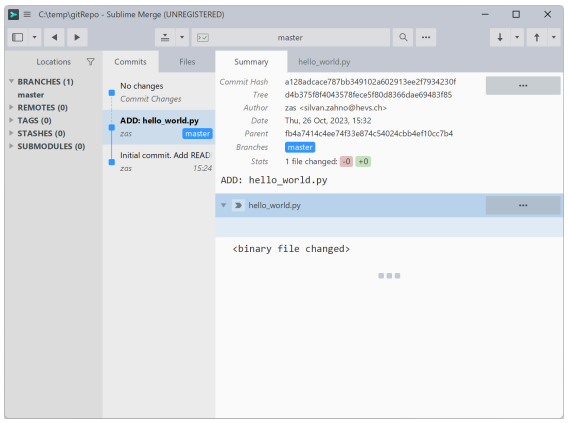
Now what new information does the command give you:

**Commandline**

```
git log
```

**GUI**

**`Select Second Commit ⇨ See all informations`**




Write down the answer!

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

```
git log --oneline
```

are only the first few characters of these so-called “short hashes”.

## 3.10 Checkout commit

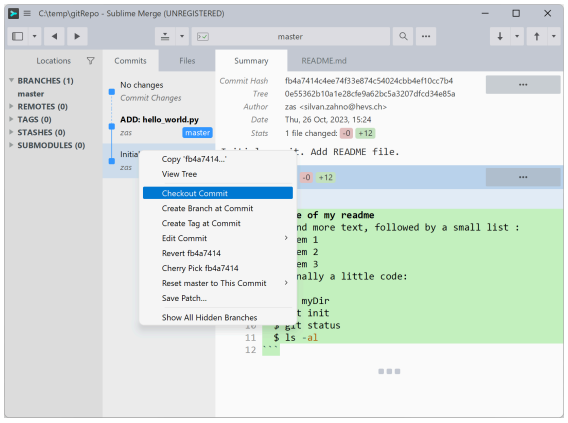
Now do a checkout with the following command using the “short hash” , which corresponds to your first commit.

**Commandline**

```
git checkout <SHA1> -b inspectingPrev
# for example
git checkout fb4a741 -b inspectingPrev
```

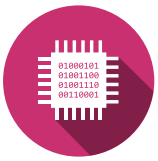
**GUI**

**`Select First Commit` ⇨ `Checkout commit`**




Now what do you notice when you call the contents of the working directory?

Write down the answer!




## 3.11 Checkout master

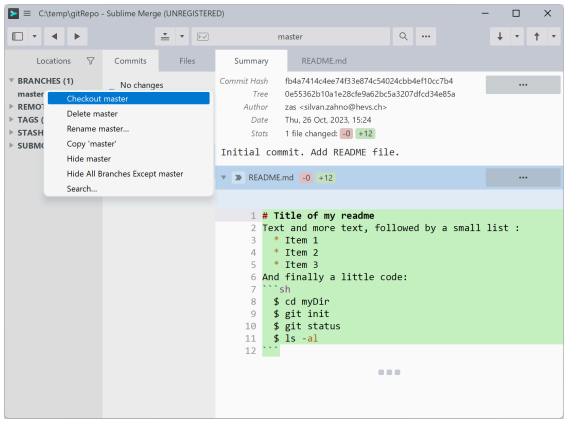
Now run a checkout with the following command:

**Commandline**

```
git checkout master
```

**GUI**

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




Now what do you notice when you look at the contents of the working directory?

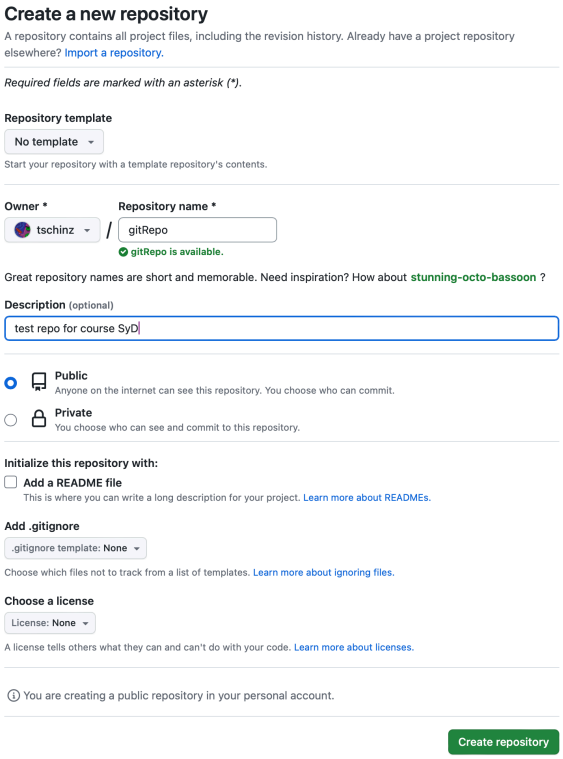
Write down the answer!


# 4 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 Github repo and add it as a remote

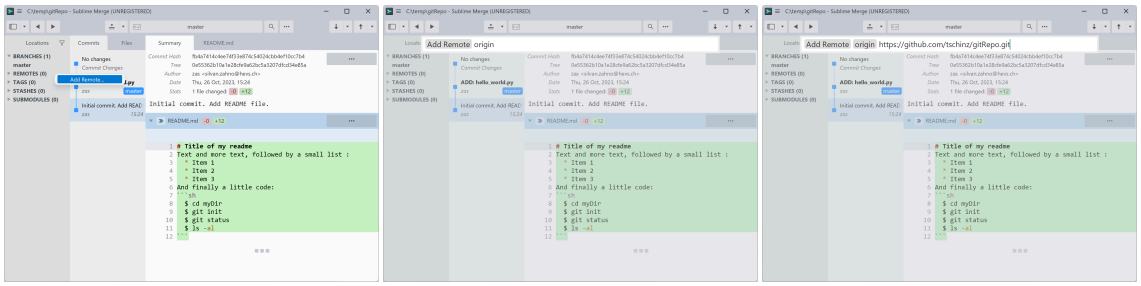



### Commandline

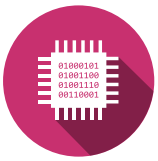
```
git remote add origin https://github.com/<username>/<reponame>.git
```

### GUI

**`Remotes (0)` ⇨ `Add Remote...` ⇨ `Remote name` ⇨ `origin` ⇨ `Remote URL` ⇨ `https://github.com/<username>/<reponame>.git`**



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



3. Create two commits on this branch:
   - One to create a file **`hello_world.py`** (This commit already exists).
   - One to populate this file **`hello_world.py`**.

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

4. Checkout the **`master`** branch
5. From the **`master`** branch, create a new development branch **`dev02`**.
6. Edit the **`README.md`** file and create a commit on the branch **`dev02`**.
7. Merge the branch **`dev02`** into **`master`**.
   - Checkout the **`master`** branch
   - Perform a merge "merge **`dev02`** into **`master`**"
8. Merge the branch **`dev01`** into **`master`**.
   - Checkout the **`master`** branch
   - Perform a merge "merge **`dev01`** into **`master`**"
9. Push your local repository to your GitHub remote.

   ```
   git push origin master
   ```

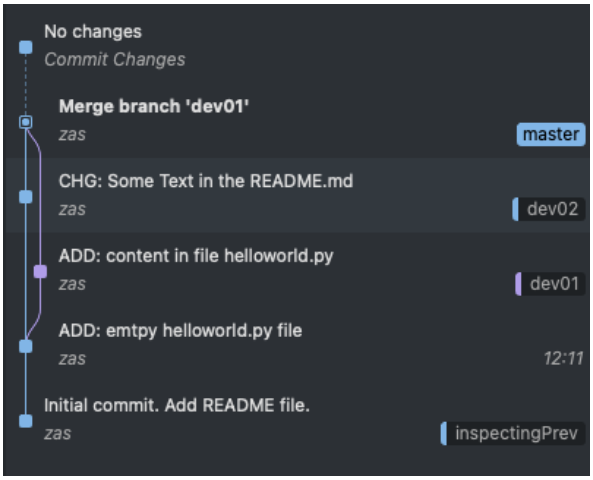



Figure 7 - Status after repo branches **`dev01`** and **`dev02`** have been merged

# 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/tschinz/gitflow [5]

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.

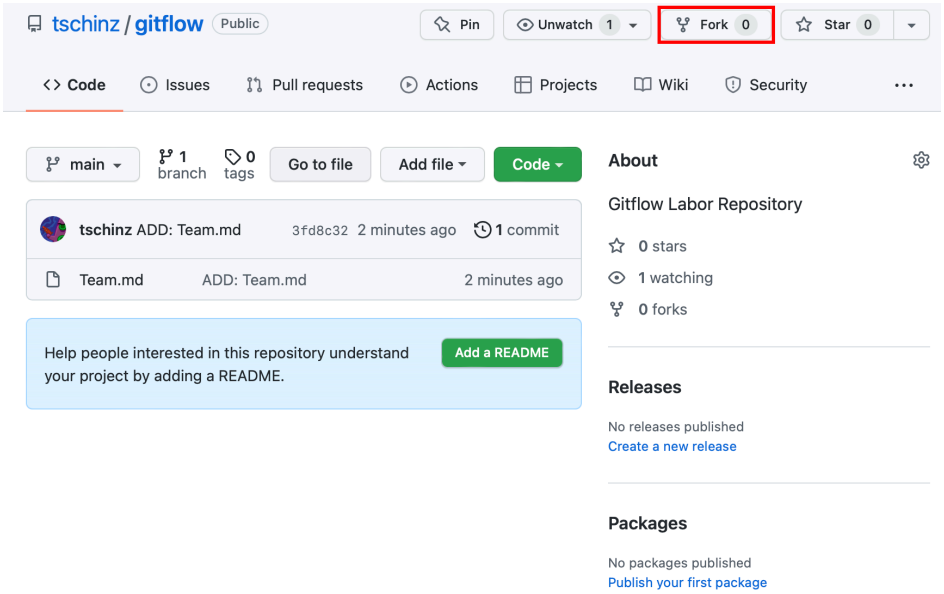



Figure 8 - Fork button for a GitHub repository

Next, clone our shared repository. The URL of your repository will look similar to:

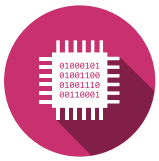
```
git clone https://github.com/<username>/gitflow.git
```

## 5.2 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.




## 5.3 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).


# 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.

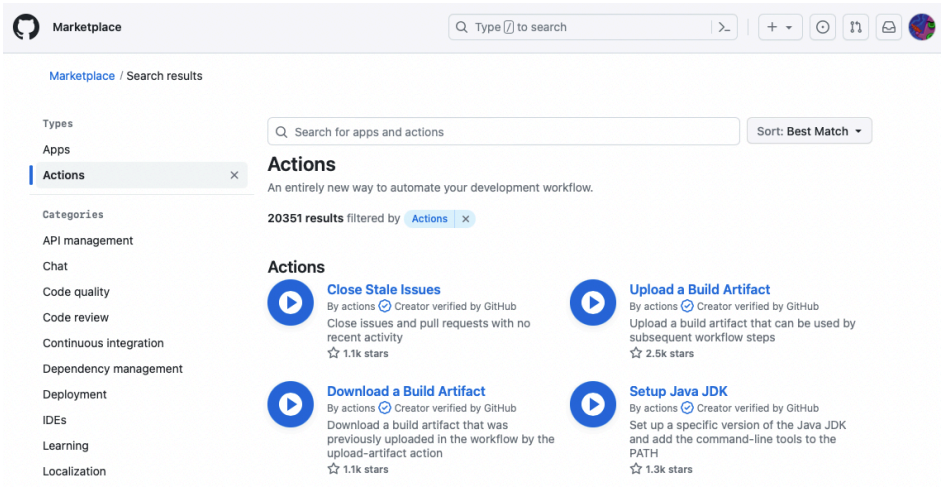



Figure 9 - Github Action Marketplace

## 6.2 Learn Git Branching

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

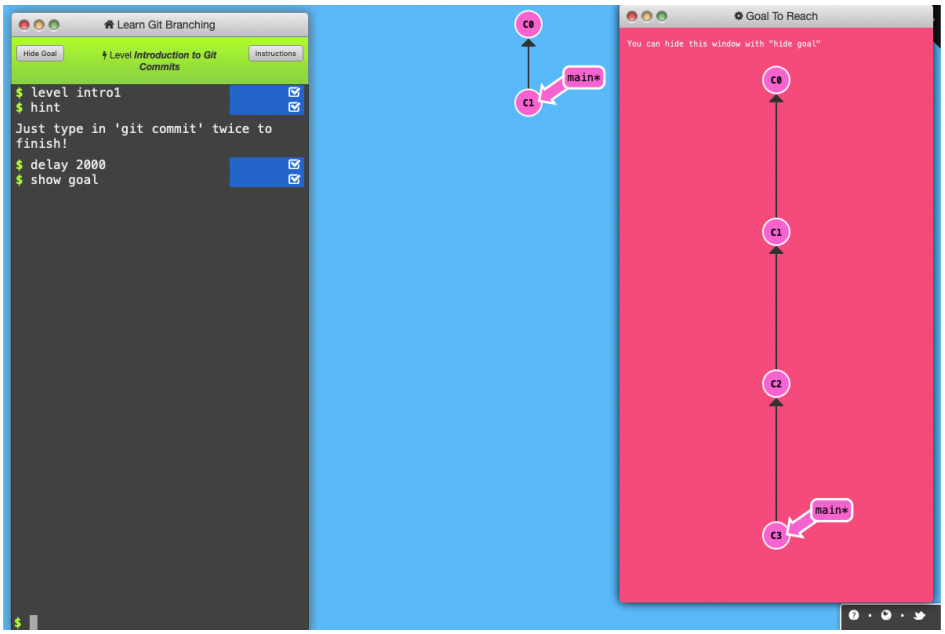
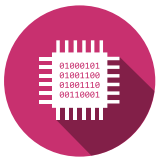



Figure 10 - Learn git branching website




# A GIT commands

Github git cheatsheet [6], [7]

## 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]
```

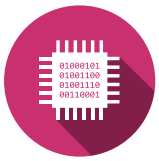
Downloads the entire history of an external repository.

```
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