



Einführung in Git - Teil A & B

Installation und Einrichtung



Inhalt

1	Ziel	3
2	Installation	4
2.1	Sublime Merge	4
2.2	git - Kommandozeile	6
2.3	Online-Konten	7
2.4	Windows-Konfiguration	7
3	Markdown	8
3.1	Markdown-Syntax	9
4	Schlusswort	10
5	Ziele	11
5.1	Tools	11
6	Grundoperationen	13
6.1	Erstellen eines git Repositories	13
6.2	Clone	14
6.3	Status abfragen	15
6.4	Stage & commit	16
6.5	Datei hinzufügen	18
6.6	Markdown	19
6.7	Online stellen	20
6.8	Zurück in der Zeit (nicht in die Zukunft)	21
6.9	Allgemeine Fragen	22
6.10	Tag	22
7	Branch und Merge	23
7.1	Erster Branch	23
7.2	Zweiter Branch	24
7.3	Merge dev02	25
7.4	Merge dev01	25
7.5	Endergebnis	26
8	Gitgraph	28
9	Gitflow	29



9.1 Fork	29
9.2 Parallele Zusammenarbeit	29
9.3 Pull Request	30
10 Extras	31
10.1 Eigenes Projekt	31
10.2 Git Branching lernen	32
Literatur	33
11 Anhang	34
A GIT Befehle	34
AA Änderungen überprüfen und eine Commit-Transaktion anfertigen	34
AB Änderungen synchronisieren	34
B Meistgebrauchten Git Befehle	36
BA Start a working area	36
BB Work on the current change	36
BC Examine the history and state	36
BD Grow, mark and tweak your common history	36
BE Collaborate	36



1 | Ziel

Dieses Labor ist in zwei Teile unterteilt. Teil A muss als Vorbereitung zu Hause durchgeführt werden, während Teil B gemeinsam im Labor bearbeitet wird. Das Labor wird eigenständig auf Ihrem Laptop durchgeführt und am Ende bewertet.

In diesem Labor werden wir die Grundprinzipien der Versionskontrolle [git](#) [1] lernen, insbesondere die Tool [Sublime Merge](#) und optional das [Git-Befehlszeilentool](#), die auf Ihrem Computer installiert und konfiguriert werden müssen (siehe Abschnitt 2). Darüber hinaus wird ein Konto auf der Plattform [Github](#) erstellt.

Schließlich lernen wir die Grundlagen von Markdown in Abschnitt 6.6, um Textdateien einfach zu schreiben.



Es ist entscheidend, dass die Installation und Konfiguration sorgfältig durchgeführt wird, um Zeitverlust während Teil B des Labors zu vermeiden.



2 | Installation

Der erste Schritt ist die Installation von Git und/oder Sublimemerge. Sie können selber auswählen ob sie die Kommandozeile oder das GUI benutzen möchten während des Labors.

2.1 Sublime Merge

Besuchen Sie die Webseite <https://www.sublimemerge.com> und laden sowie installieren Sie das Tool Sublime Merge herunter [2].

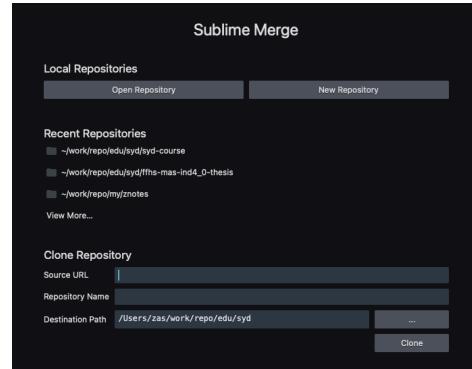


Abbildung 1 - Sublime Merge GUI

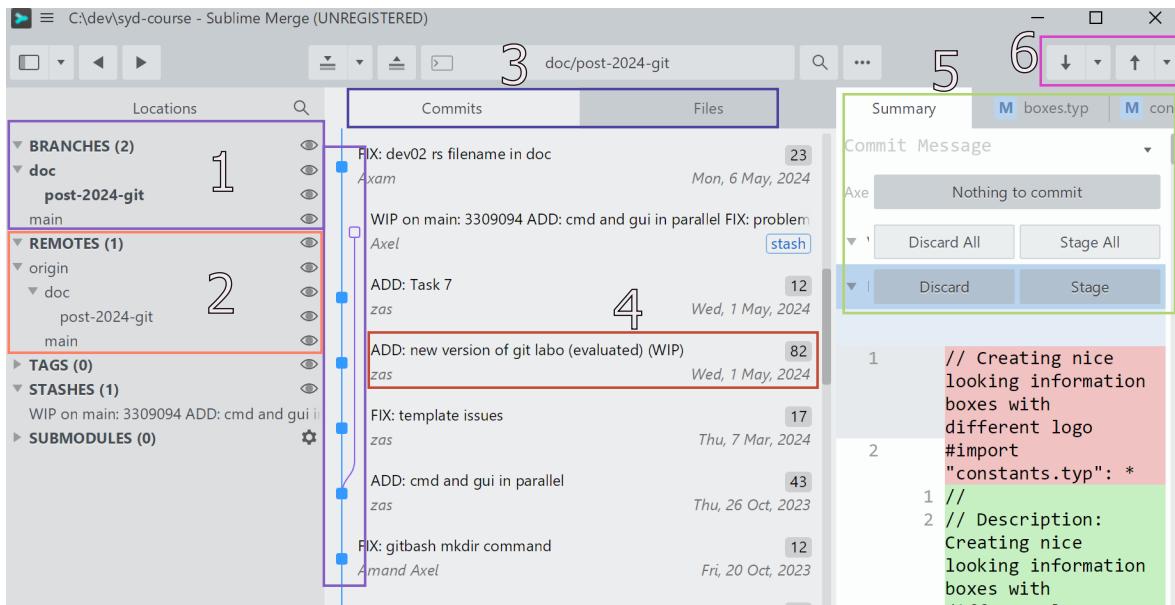
2.1.1 Konfiguration

Beim Klonen eines Repositories wird Sublime Merge automatisch nach deiner Identität fragen und dich auffordern, dich bei deinem Github-Konto anzumelden.



2.1.2 Übersicht

Die Oberfläche von Sublime Merge ist in der Abbildung 2 dargestellt:





2.2 git - Kommandozeile

Du kannst die neueste Version über die offizielle Website <https://git-scm.com/> [1] herunterladen. Git ist für Linux, Mac und Windows verfügbar. Für dieses Labor wird git ≥ 2.27 benötigt.

Starte "Git Bash" auf Windows der „Terminal“ auf MacOS. Dies ist ein Unix/Linux-ähnlicher Befehlseditor, der es ermöglicht, Git-Befehle im Konsolenmodus auszuführen.

```

Last login: Tue Mar  8 09:26:26 on ttys004
[~] zac@zac: ~ [base]
$ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
       [--exec-path[=<path>]] [(--cd)-path] [(--man-path[=<path>]]
       [(--info-path[=<path>]] [(--repository-path[=<path>]] [(--bare)]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       [--super-prefix=<path>] [--config-env=<name>] [<envvar>]
       [<command>] [<args>]

These are common Git commands used in various situations:
start a working area (see also: git help tutorial)
clone   Clone a repository into a new directory
init    Create an empty Git repository or reinitialize an existing one
work on the current change (see also: git help everyday)
add     Add files from the working tree to the index
mv     Move or rename a file, a directory, or a symlink
restore Restore working tree files
rm     Remove files from the working tree and from the index
examine the history and state (see also: git help revisions)
blame   Show which commit last modified a line of code that introduced a bug
diff    Show changes between commits, commit and working tree, etc
grep    Print lines matching a pattern
log    Show commit log messages
show   Show various types of objects
status  Show the working tree status
grow, mark and tweak your common history
branch List, create, or delete branches
commit Record changes to the repository
merge  Join two or more development histories together
rebase  Reapply commits on top of another base tip
stash  Store changes HEAD to the specified state
switch Switch branches
tag    Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
fetch  Download objects and refs from another repository
pull   Fetch and merge new commits from another repository or a local branch
push   Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

[~] zac@zac: ~ [base]
$ -3

```

Abbildung 3 - git Terminal

Beachten Sie, dass Sie für alle Befehle in Git Bash Hilfe erhalten können, indem Sie **--help** nach dem Befehl einfügen.



`git --help`

2.2.1 Globale Konfiguration

Eine Vielzahl von Einstellungen kann in Git konfiguriert werden. Es ist möglich, die Einstellungen global auf deinem Computer (Flag **--global**) oder nur für ein bestimmtes Repository zu ändern.

Wir werden nun die Minimalkonfiguration durchführen. Verwende die folgenden Befehle, um deine Identität in Git **global** auf dem System einzustellen. Verwende deinen Namen und deine E-Mail-Adresse. Diese Informationen sind öffentlich sichtbar, um deine Arbeit (deine Commits) zu identifizieren.

```

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

```

Zum Beispiel:

```

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

```

Sie können die Konfiguration mit dem folgenden Befehl überprüfen:



```
git config --list
```

Sie können auch eine bestimmte Einstellung überprüfen:

```
git config user.name
```

2.3 Online-Konten

2.3.1 Github

Besuchen Sie die Webseite <https://github.com> und erstellen Sie ein Konto und loggen Sie sich ein.

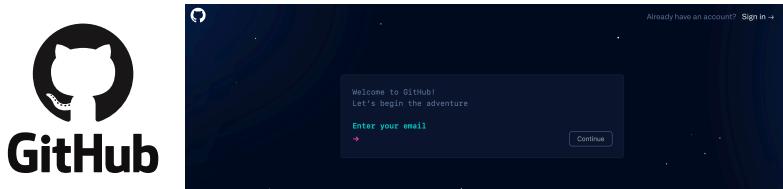


Abbildung 4 - GitHub Anmeldung

2.4 Windows-Konfiguration

Um auch den versteckten `.git`-Ordner sowie die Dateierweiterungen sehen zu können. Konfigurieren Sie Ihren Windows Datei Explorer wie folgt:

Datei-Explorer \Rightarrow Ansicht \Rightarrow Anzeigen \Rightarrow Aktivieren Sie „**Dateinamenerweiterungen**“ und „**Versteckte Elemente**“

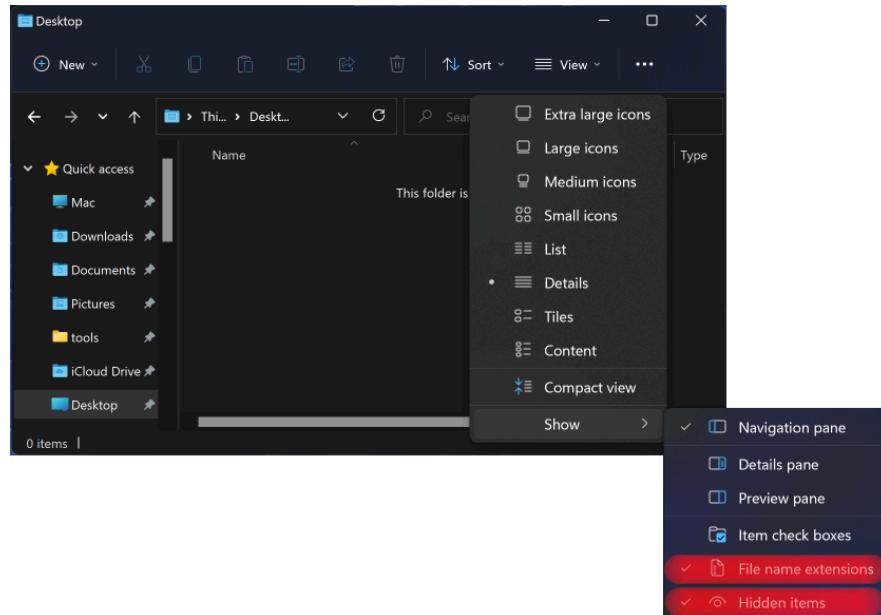


Abbildung 5 - Windows Datei Explorer Konfiguration



3 | Markdown

Markdown ist eine leichte Beschreibungssprache mit einer Syntax zur Formatierung von Klartext. Sie ist so konzipiert, dass sie leicht zu lesen und zu schreiben ist und gleichzeitig leicht in PDF, HTML oder andere Formate konvertiert werden kann. Markdown wird häufig für die Formatierung von Text im Web verwendet, z. B. in **README.md**-Dateien, Dokumentationen, Forenbeiträgen und Nachrichten.

Um Markdown zu schreiben, benötigen Sie Ihren bevorzugten Texteditor oder Sie können einen spezialisierten Markdown-Editor wie **Marktext** installieren.

Zum Beispiel ist die [Einführungsseite](#) dieses Kurses in Markdown geschrieben. Sie können den Quellcode der Seite sehen, indem Sie auf die Schaltfläche „Diese Seite bearbeiten“ in der oberen rechten Ecke der Seite oder über den [link](#) klicken.

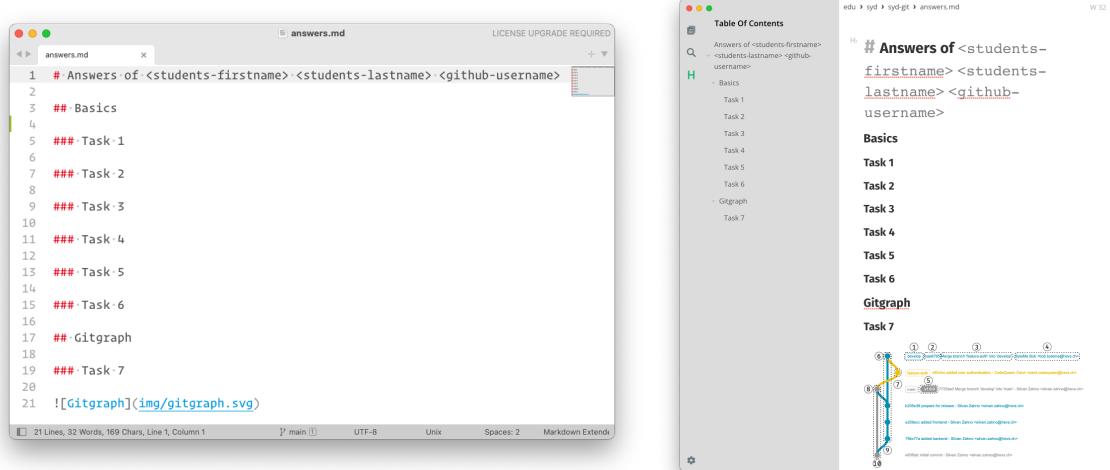


Tabelle 1 - Links: Einfacher Texteditor (Sublime Text), Rechts: Marktext



Für das Labor müssen Sie ein Dokument im Markdown-Format schreiben.
Halten Sie Ihren Editor bereit.



3.1 Markdown-Syntax

Nachfolgend ein kurzer Überblick darüber, wie eine Markdown-Datei aufgebaut ist. Die Syntax ist einfach und leicht zu erlernen. Die Datei muss mit der Endung `.md` gespeichert werden. Eine vollständigere Syntaxliste finden Sie unter https://wiki.zahno.dev/multimedia/writing/md/md_github.html.

```
# Title 1
## Title 2
### Title 3

Some simple Text italic bold
~~Strikethough~~ `monospaced`

Formulas  $S = \sum_{i=1}^n x_i^2$ 

- 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

```

urces > git > markdown-cheatsheet.md W 90

Title 1

Title 2

Title 3

Some simple Text *italic* **bold** Strikethough monospaced

Formulas $S = \sum_{i=1}^n x_i^2$

- List Item 1
- List Item 2

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

[Link name](#)



```
// 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 | Schlusswort

Herzlichen Glückwunsch! Sie haben nun alles installiert und konfiguriert, was Sie benötigen, um mit Git zu arbeiten. Sie sollten Folgendes getan haben:

- Git und Sublime Merge installiert
- Git mit Ihrem Namen und Ihrer E-Mail-Adresse konfiguriert
- Ein GitHub-Konto erstellt
- Mit der grafischen Benutzeroberfläche (GUI) von Sublime Merge vertraut gemacht
- Mit der Theorie und den Befehlen von Git vertraut gemacht
- Mit Markdown und seiner Syntax vertraut gemacht



Im Anhang Abschnitt A und Abschnitt B finden Sie eine Zusammenfassung der wichtigsten Git Befehle.



5 | Ziele

In diesem Labor lernen wir die Grundprinzipien von der Versionskontrolle **git** kennen [1]. Sie müssen bereits Teil A des Labors zuhause durchgeführt haben um mit dem Teil B beginnen zu können.

Im Abschnitt 6 lernen wir die Basis Operationen kennen um mit Git arbeiten zu können. Das erstelle Repository wird danach auf [GitHub](#) veröffentlicht. Die erweiterten Funktionen **branch** sowie **merge** werden in einem Beispiel probiert im Abschnitt 7. Im Abschnitt 9 arbeiten wir alle gemeinsam an einem Repository. Schlussendlich gibt es einige optionale Arbeiten im Abschnitt 10.



Die Antworten auf die Fragen **müssen** in der Markdown-Datei **answer.md** niedergeschrieben werden. Die Datei befindet sich im Repository das im nächsten Schritt erstellen wird.



Stellen Sie am Ende des Labors sicher dass Sie alle Änderungen auf GitHub veröffentlicht haben. Nur die auf GitHub veröffentlichten Änderungen werden bewertet.

5.1 Tools

In diesem Labor werden Sie Sublime Merge als Grafisches Tool und/oder Git CMD als Kommandozeile benutzen.



Git CMD
App



Sublime Merge
App

Abbildung 6 - Git CMD Kommandozeile Abbildung 7 - Sublime Merge GUI



Benutzen Sie den Prozess in der Kommandozeile nur wenn Sie sich mit einem Terminal wohlfühlen. Sie müssen die Befehle kennen um navigieren, Informationen anzeigen etc. **cd**, **ls**, **cat**, **touch**, **nano** ...



5.1.1 Übersicht

Die Oberfläche von Sublime Merge ist in der Abbildung 8 dargestellt:

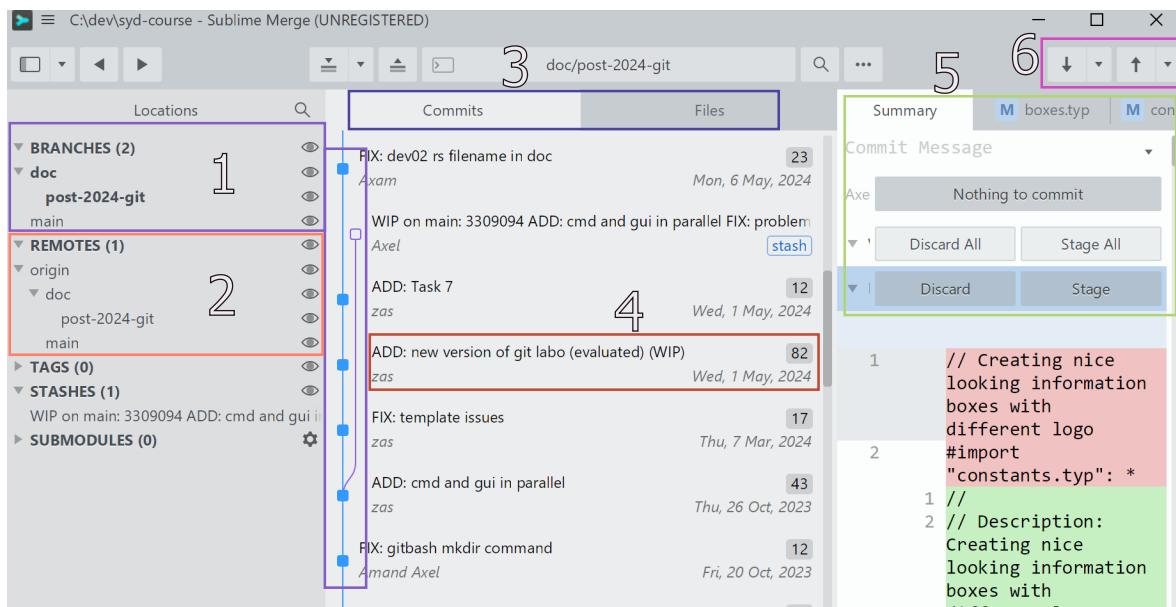


Abbildung 8 - Sublime Merge GUI

- Branches:** Liste der Branches des Repositories. Du kannst Branches erstellen, löschen und umbenennen. Sie werden auch in ihrer Chronologie unter dem Tab **Commits** (Punkt 3) angezeigt.
- Remotes:** Liste der Remote-Repositories, d.h. die Server, auf denen der Code gespeichert ist und auf die du **pushen** kannst.
- Commits/Files:** Tabs, um zwischen der chronologischen Ansicht der Commits und der Ansicht der Änderungen für den ausgewählten Commit zu wechseln.
- Commit-Beschreibung:** Ein Commit besteht aus einer Nachricht, einem Autor und einem Datum.
- Aktuelle Änderungen:** Die Dateien, die im Vergleich zum letzten Commit geändert wurden, werden hier aufgelistet. Du kannst die Dateien auswählen, die du **committed** möchtest, indem du sie auswählst und auf die Schaltfläche **Stage** klickst. Solange die Dateien nicht **committed** sind, kannst du eine Datei auch wieder aus dem Staging entfernen, indem du auf die Schaltfläche **Unstage** klickst oder die Änderungen einer Datei vollständig löschen, indem du auf **Discard** drückst (⚠️ endgültige Löschung ⚠️).
- Pull / Push:** Die beiden Schaltflächen ermöglichen es, **pull** - die neuesten Änderungen vom Remote-Repository zu übernehmen - und **push** - die lokalen Änderungen an das Remote-Repository zu senden. Es ist auch möglich, **fetch** zu verwenden, indem du auf den kleinen Pfeil neben der Schaltfläche **Pull** klickst, was es dir ermöglicht, die neuen Commits zu sehen, ohne das lokale Repository zu ändern.



6 | Grundoperationen

6.1 Erstellen eines git Repositories

Um auf einer gemeinsamen Basis zu arbeiten, werden wir eine Kopie des Vorlagen-Repositories erstellen. Diese Operation wird **fork** genannt - *Abbildung 9*. Die vorgenommenen Änderungen werden in Ihrem eigenen Repository widergespiegelt, nicht in dem, das kopiert wurde. So können Sie arbeiten, ohne Synchronisationsprobleme zu befürchten.

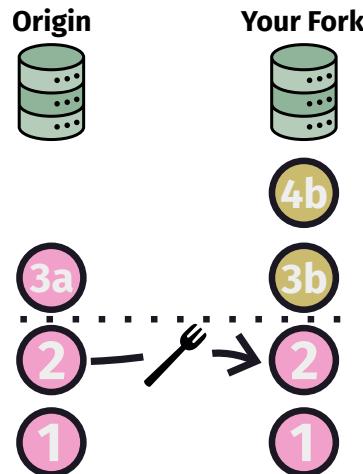


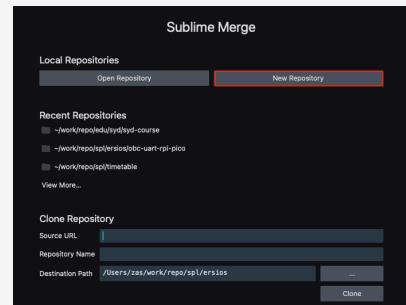
Abbildung 9 - Fork des Vorlagen-Repositories

Wenn Sie ein neues Repo von Grund auf neu erstellen möchten (nicht nötig in diesem Labor), können Sie dies mit dem folgenden Befehl oder über das GUI tun:



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

Datei ⇒ Neues Repository



Das neu erstellte Repo wird leer sein, ohne **remote**, d.h. ohne Verbindung zu einem entfernten Repo (z.B. GitHub). Sie können später ein entferntes Repo mit dem Befehl **git remote add origin <url>** oder über das GUI hinzufügen.



Erstellen Sie einen Fork des Vorlagen-Repositories mit dem folgenden Link
<https://classroom.github.com/a/rYLfVeC0>.



Dazu müssen Sie sich bei GitHub anmelden. Akzeptieren Sie die Classroom-Einladung - *Abbildung 11*, die Ihnen den Link zu IHREM eigenen Repository gibt - *Abbildung 12*.

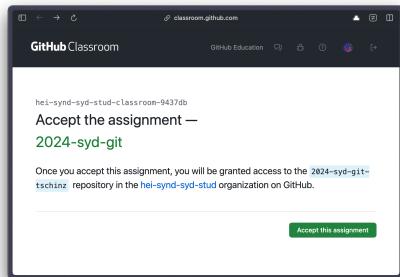


Abbildung 11 - Einladungslink zum Forken

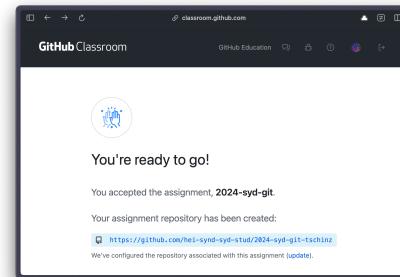


Abbildung 12 - Link zum Klonen des Repos



Notieren Sie den Link zu Ihrem Repository für den nächsten Schritt.
Nutzen Sie die Gelegenheit, um Ihr Repo online unter dem angegebenen Link zu besuchen, indem Sie das `.git` am Ende der URL entfernen.

6.2 Clone

Nachdem Sie den Fork erstellt haben, erhalten Sie einen Link auf Ihr eigenes Repository - *Abbildung 12*. klonen Sie es auf Ihren lokalen Computer an einem von Ihnen bevorzugten Speicherort.

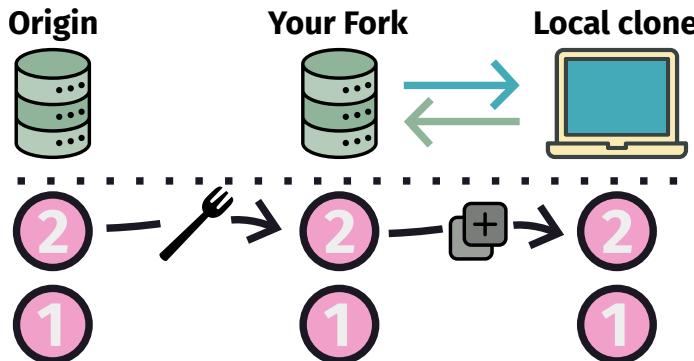


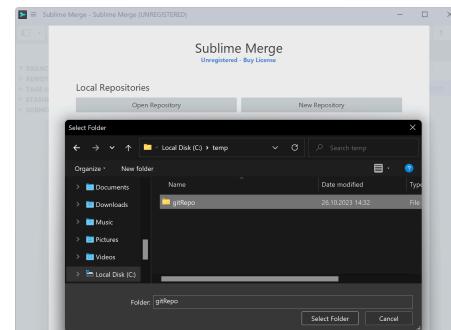
Abbildung 13 - Klonen

Commandline

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

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

GUI - CTRL+T ⇒ Paste Source URL ⇒ Select Folder





6.3 Status abfragen

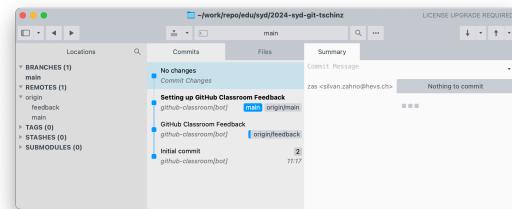
Erhalten Sie Informationen über Ihr Repo mit den folgenden Befehlen:

Commandline

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

GUI

See the status in the main window.



Aktuell hat Ihr Repo keine Änderungen, da nichts geändert wurde. In Sublime Merge steht oben im Fenster **No changes** (Keine Änderungen).



6.4 Stage & commit

Wir werden nun Daten im Repo ändern, in diesem Fall die Datei `answers.md`.

Aufgabe 0

Öffnen Sie das Dokument `answers.md` und ersetzen Sie:



- `<students-firstname>` mit Ihrem Vornamen
- `<students-lastname>` mit Ihrem Nachnamen
- `<github-username>` mit Ihrem GitHub Benutzernamen

Nachfolgend müssen die Abschnitte **Aufgaben** in dieser Datei `answers.md` beantwortet werden.

Wie unter Abschnitt 6.3 dargestellt, überprüfen Sie den Status Ihres Repositories.



Aufgabe 1

Was ist der Status? Was bedeutet es?

Wir werden die Änderungen lokal speichern. Dazu ist es notwendig, die Änderungen zu **Committen**.

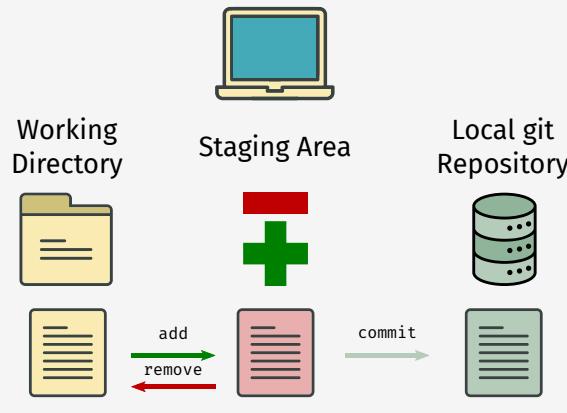


Abbildung 14 - Arten von lokalen Git Operationen

Ihr lokales Git-Repo besteht aus drei Bereichen, die von git gepflegt werden:

- Das Working directory ist ein Verzeichnis, das die aktuelle Version deiner Dateien enthält (in den Augen deines Betriebssystems ein normales Dateiverzeichnis)
- Stage enthält die Änderungen, die in den nächsten Commit aufgenommen werden sollen;
- Der Head zeigt auf den Ort im Git-Repo-Baum, an dem der nächste Commit durchgeführt werden soll.



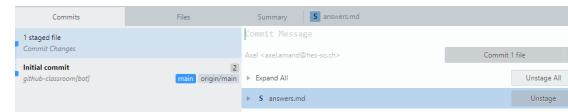
Um die Änderungen *lokal* zu speichern, müssen Sie zunächst die Änderungen auswählen, die im Repo gespeichert werden sollen. Dieser Prozess wird **Stage** genannt.

Commandline

```
git add answers.md
```

GUI

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



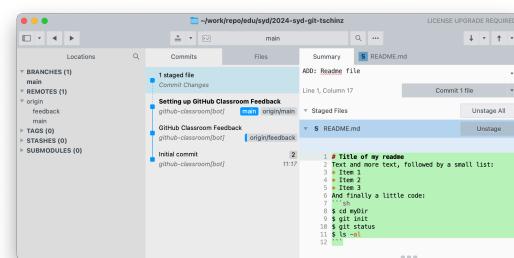
Sobald die Änderungen ausgewählt wurden, wird der Commit durchgeführt. Ein Commit ist ein Schnappschuss Ihres Repos zu einem bestimmten Zeitpunkt und sobald er durchgeführt wurde, wird der Zustand des Ordners im Repository gespeichert. Der **Commit** wird durch eine eindeutige Kennung (Hash) identifiziert und enthält Informationen über den Autor, das Datum und die Commit-Nachricht, die die vorgenommenen Änderungen widerspiegeln sollte:

Commandline

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

GUI

Commit Message ⇒ CHG: personal data in answers.md ⇒ **Commit 1 file**



Aufgabe 2

Nachdem Sie einen Commit wie oben beschrieben durchgeführt haben, beantworten Sie die folgenden Fragen:



- Was bedeutet die Beispiel-Commit-Nachricht?
- Ist es möglich, einen Commit ohne Nachricht zu erstellen?
- Was ändert sich jetzt im Repo?
- Ist das Remote-Repo (hier Github) mit unseren Änderungen aktuell?



6.5 Datei hinzufügen

Erstellen Sie dann eine leere Datei mit dem Namen **README.md** im Hauptverzeichnis Ihres Repos:

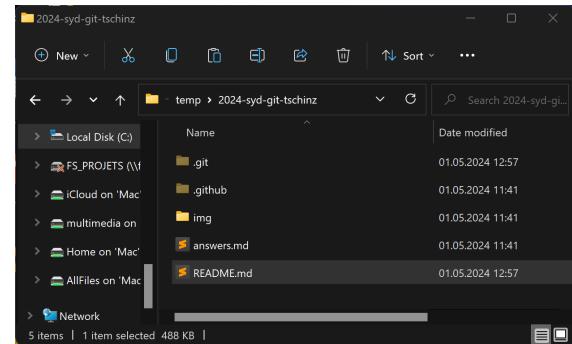
Commandline

```
touch README.md
```

GUI

(Windows) **C:\path\to\repo** ⇒ **New** ⇒ **Any File** ⇒ **README.md**

(Linux&MacOS) **/path/to/repo** ⇒ **New** ⇒ **Any File** ⇒ **README.md**



Aufgabe 3



Schauen Sie sich noch einmal den Status Ihres Repos an, was haben Sie festgestellt?



Machen Sie einen Commit mit der neuen Datei sowie den Antworten auf die Aufgaben 2 und 3.

Beispiel-Repo

Ein einfaches Git-Repo, das aus fünf Commits besteht, kann folgendermassen dargestellt werden. Die Position **Head** ist ein Verweis auf einen Commit, der den aktuellen Status/die aktuelle Ansicht des Repos darstellt, hier die letzte Änderung:

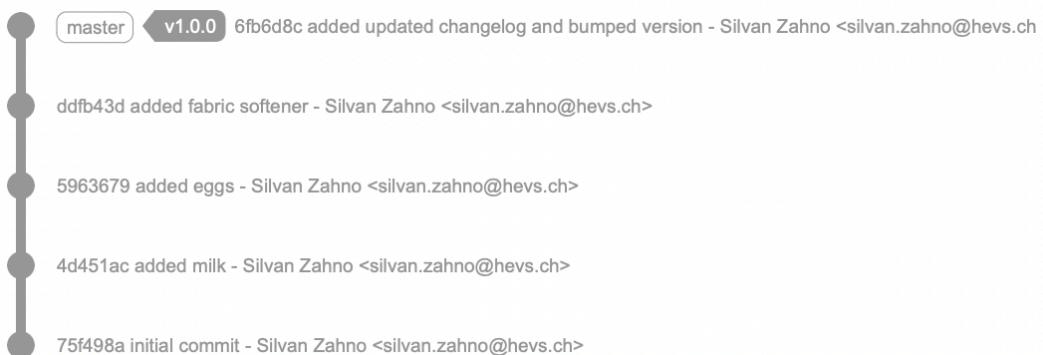


Abbildung 15 - Fünf commits auf dem lokalen Repo, jeder commit besitzt seine eigene identifikation



6.6 Markdown

Ändern Sie die Datei **README.md** mit einem Texteditor, damit sie ungefähr so aussieht wie die folgende:

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

Abbildung 16 - Markdown-Struktur zu reproduzieren

Um Ihnen zu helfen, können Sie Websites wie <https://markdownlivepreview.com/> verwenden, um das Ergebnis anzuzeigen.

Hier ist eine Cheatsheet Markdown, die Ihnen hilft, Ihre Datei **README.md** zu erstellen:

```
# 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
```



Vervollständigen Sie Ihre Datei **README.md**. Wenn Sie zufrieden sind, committen Sie Ihre Änderungen.



## 6.7 Online stellen

Derzeit wird die gesamte Arbeit nur *lokal* gespeichert. Um sie online zu stellen, ist es notwendig, die Commits auf das Remote-Repo (Github) zu **pushen**, um eine Kopie davon zu haben und anderen im Rahmen einer Gruppenarbeit zu ermöglichen, sie zu sehen:

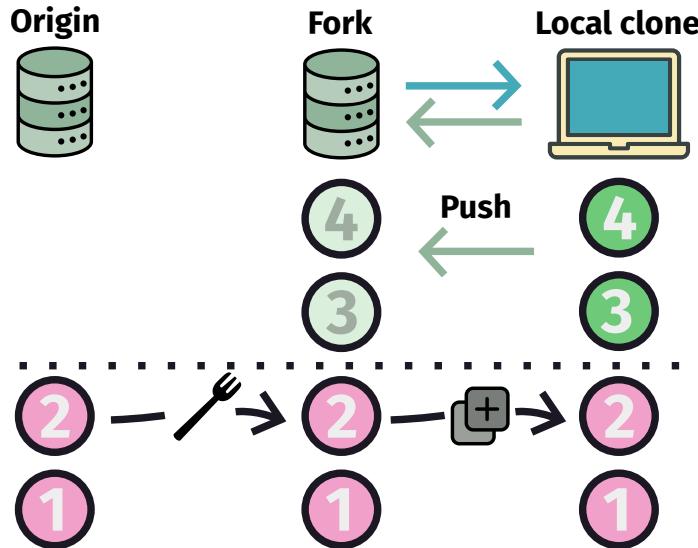


Abbildung 17 - Push auf das Remote-Repository

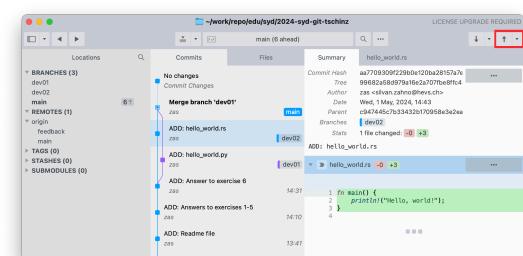
Um dies zu tun, nachdem alle Änderungen committet wurden:

**Commandline**

```
git push origin main
```

**GUI**

Top Right Arrow  $\Rightarrow$  Push changes



Gehen Sie online zu Github, um das Ergebnis Ihrer Arbeit zu sehen.  
Sie sollten die Datei **README.md** mit dem von Ihnen erstellten Inhalt in lesbarer Form sehen.



## 6.8 Zurück in der Zeit (nicht in die Zukunft)

Git ist ein Versionsverwaltungssystem. Es ist also möglich, zu einer früheren Version des Repositories zurückzukehren. Dazu müssen Sie zunächst den Commit identifizieren, zu dem Sie zurückkehren möchten. Hier kehren wir zum allerersten Commit zurück.

Jeder Commit ist mit einem “Hash” oder einer “Prüfziffer” (vom Typ **sha1**) versehen. Dieser Hash ist eine eindeutige Kennung, die es ermöglicht, einen bestimmten Commit zu isolieren. Letzteres ist sichtbar, indem Sie auf einen Commit für SublimeMerge klicken oder den folgenden Befehl verwenden:

```
git log --oneline
```

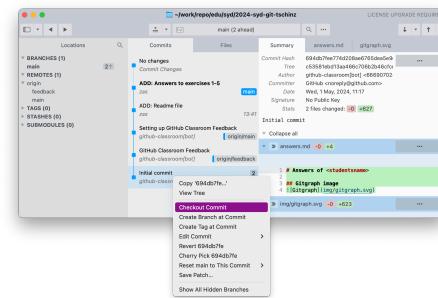
Um zu einem Commit zu wechseln, führen Sie die **checkout**-Operation aus:

### Commandline

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

### GUI

**Select First Commit (Initial Commit) ⇒ Right click ⇒ Checkout commit**



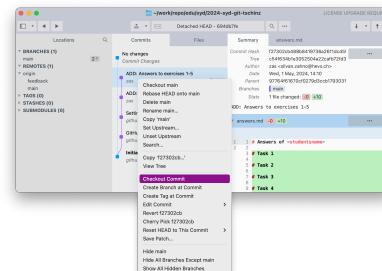
Kehren Sie dann auf die gleiche Weise zum letzten Commit zurück. Abgesehen vom Hash ist es möglich, zum letzten Commit zurückzukehren, indem Sie den Namen des Branches verwenden, hier **main**.

### Commandline

```
git checkout main
```

### GUI

**Branches (1) ⇒ main ⇒ checkout main**



### Aufgabe 4

Was passiert, wenn Sie zum Commit „Initial commit“ zurückkehren? Was passiert, wenn Sie zum letzten Commit zurückkehren?



## 6.9 Allgemeine Fragen

### Aufgabe 5



Was ist der Unterschied zwischen dem lokalen Repository und dem Remote-Repository?

Was würde passiert, wenn Sie das lokale Repository löschen?

### Aufgabe 6



Was ist mit dem [ursprünglichen Repository](#), das geforkt wurde?

Wurde es geändert?

## 6.10 Tag

Um Schlüssel-Commits zu kennzeichnen, können Sie **tags** erstellen. Ein Tag ist ein Zeiger auf einen bestimmten Commit. Es wird häufig verwendet, um Codeversionen zu kennzeichnen, z.B. **v1.0**, **v2.0**, usw.

Für den Abschluss dieses Kapitels erstellen Sie ein Tag **chapter2** auf dem letzten Commit. Dazu:

### Commandline

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

### GUI

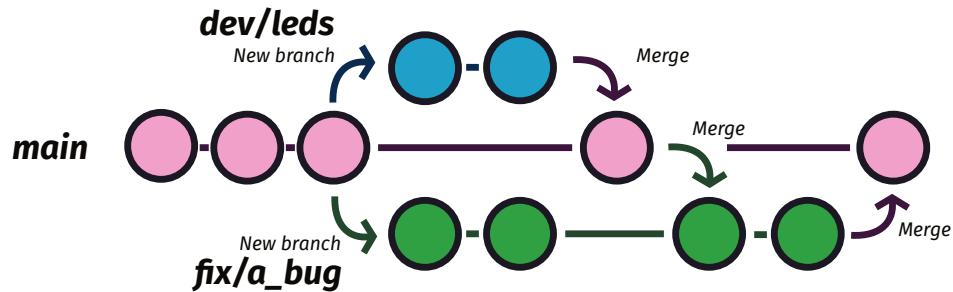
- Right click on commit ⇒ Create Tag at Commit ⇒ chapter2 ⇒ Enter
- Right click on tag ⇒ Tag chapter2 ⇒ Push Tag chapter2



# 7 | Branch und Merge

Bisher haben wir die grundlegenden Funktionen von Git verwendet - [rosa Commit-Linie](#).

Es gibt auch die Funktionen **branches** und **merge**, die Git im Vergleich zu früheren Tools stark vereinfacht hat:



Branches ermöglichen es, parallel an mehreren Versionen eines Projekts zu arbeiten. Zum Beispiel können Sie einen **Branch erstellen**, um eine neue Funktion zu entwickeln, während Ihr Kollege an einem **Patch arbeitet**, um einen Fehler zu beheben. Das Zusammenführen (**merge**) ermöglicht es, Änderungen von einem Branch in einen anderen zu integrieren.



Git ist nicht allwissend. Wenn zwei Branches dieselben Dateien geändert haben, weiß Git nicht, wie sie zusammengeführt werden sollen, und Sie müssen die Konflikte manuell lösen.

## 7.1 Erster Branch

Der erste Branch simuliert die Arbeit einer ersten Person an einer neuen Datei, `hello_world.py`. Dies könnte jede Art von Datei sein: Code, Bild, Inventor-Zeichnung, Matlab-Simulation ...

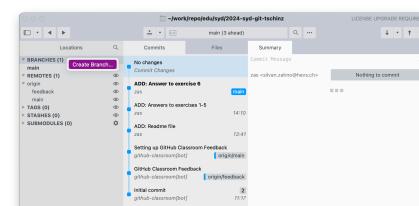
1. Erstellen Sie einen Entwicklungs-Branch `dev01` in Ihrem lokalen Repository:

### Befehlszeile

```
git checkout -b dev01
```

### GUI

**Branches**  $\Rightarrow$  **Create Branch**  $\Rightarrow$  `dev01`



2. Erstellen Sie einen Commit in diesem Branch:

- Erstellen Sie eine Datei `hello_world.py`
- Füllen Sie sie mit folgendem Code:

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

- Führen Sie einen Commit durch, wie im Kapitel Abschnitt 6.4 beschrieben, mit einer klaren Nachricht.



## 7.2 Zweiter Branch

Der zweite Branch simuliert die Arbeit einer zweiten Person an einer anderen Datei, `hello_world.rs`.

Angenommen, der Branch wurde gleichzeitig mit dem anderen erstellt, ist es notwendig, zunächst zum gleichen Commit wie zuvor zurückzukehren.

1. Wechseln Sie zurück zum Branch `main`:

### Befehlszeile

```
git checkout main
```

### GUI

`main` ⇒ Rechtsklick ⇒ Checkout `main`



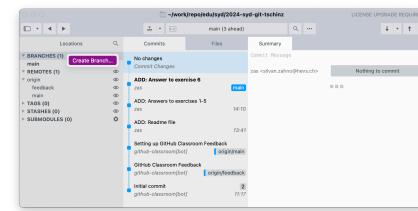
2. Erstellen Sie von `main` aus einen neuen Entwicklungs-Branch `dev02`:

### Befehlszeile

```
git checkout -b dev02
```

### GUI

Branches ⇒ Create Branch ⇒ `dev02`



3. Erstellen Sie einen Commit in diesem Branch:

- Erstellen Sie eine Datei `hello_world.rs`
- Füllen Sie sie mit folgendem Code:

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

- Führen Sie einen Commit durch, wie im Kapitel Abschnitt 6.4 beschrieben, mit einer klaren Nachricht.



### 7.3 Merge dev02

Nachdem beide Branches Commits erhalten haben, ist es Zeit, sie in den Haupt-Branch **main** zu integrieren.

1. Wechseln Sie zurück zum Branch **main**:

#### Befehlszeile

```
git checkout main
```

#### GUI

**main** ⇒ Rechtsklick ⇒ Checkout **main**



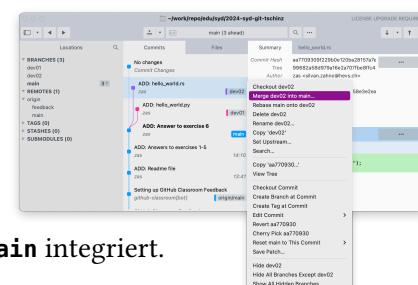
2. Führen Sie den Merge von Branch **dev02** in **main** durch:

#### Befehlszeile

```
git merge dev02
```

#### GUI

**Commit** ⇒ Merge **dev02** into **main** ...



Die Arbeit des Branches **dev02** ist jetzt in den Branch **main** integriert.

### 7.4 Merge dev01

Wie zuvor ist es jetzt an der Zeit, den Branch **dev01** in **main** zu integrieren.

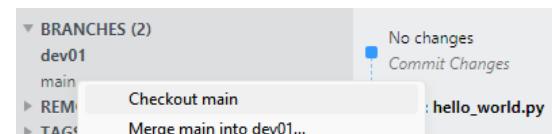
1. Stellen Sie sicher, dass Sie sich im Branch **main** befinden:

#### Befehlszeile

```
git checkout main
```

#### GUI

**main** ⇒ Rechtsklick ⇒ Checkout **main**



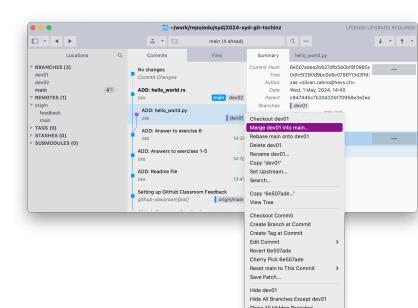
2. Führen Sie den Merge von Branch **dev01** in **main** durch:

#### Befehlszeile

```
git merge dev01
```

#### GUI

**Commit** ⇒ Merge **dev01** into **main** ...





## 7.5 Endergebnis

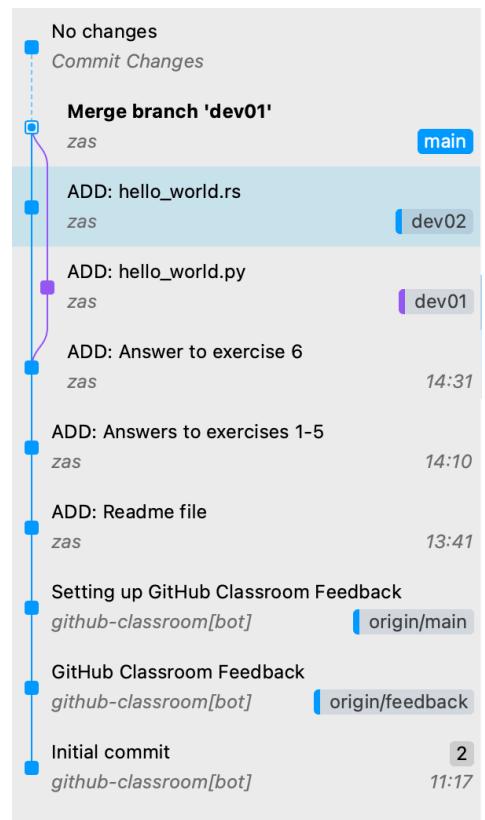


Abbildung 27 - Status nach der Zusammenführung der Repository-Zweige **dev01** und **dev02**

Es kann sein, dass das Repository standardmäßig nur eine blaue Linie anzeigt. Da die Arbeit sehr linear ist, verbirgt Sublime Merge die Commits. Sie können



die Anzeige aller Commits erzwingen, indem Sie auf die Schaltfläche  in der Lebenslinie der Commits klicken.

- Pushen Sie Ihr Repository auf den entfernten GitHub-Server. **Es sind drei push-Befehle erforderlich, um jeden Branch separat zu pushen:**

### Befehlszeile

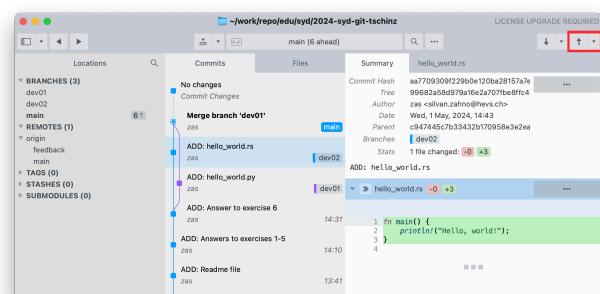
```
git push origin main
git push origin dev01
git push origin dev02
```

### GUI

**Checkout dev01**  $\Rightarrow$  Oben Rechts Pfeil  $\Rightarrow$  Änderungen pushen

**Checkout dev02**  $\Rightarrow$  Oben Rechts Pfeil  $\Rightarrow$  Änderungen pushen

**Checkout main**  $\Rightarrow$  Oben Rechts Pfeil  $\Rightarrow$  Änderungen pushen





2. Taggen Sie den Branch **main** mit dem Tag **chapter3**, wie im Kapitel Abschnitt 6.10 beschrieben.  
Vergessen Sie nicht, den Tag zu pushen!



Der Status Ihres Repositories sollte ähnlich wie in Abbildung 27 + Tags **chapter2** sowie **chapter3** aussehen. Dies ist Teil der Bewertung.



## 8 | Gitgraph

In der Abbildung 29 ist ein Beispiel eines Git Repositories dargestellt. Benennen Sie alle Elemente die im auf dem Bild zu erkennen sind (Punkte 1- 10).

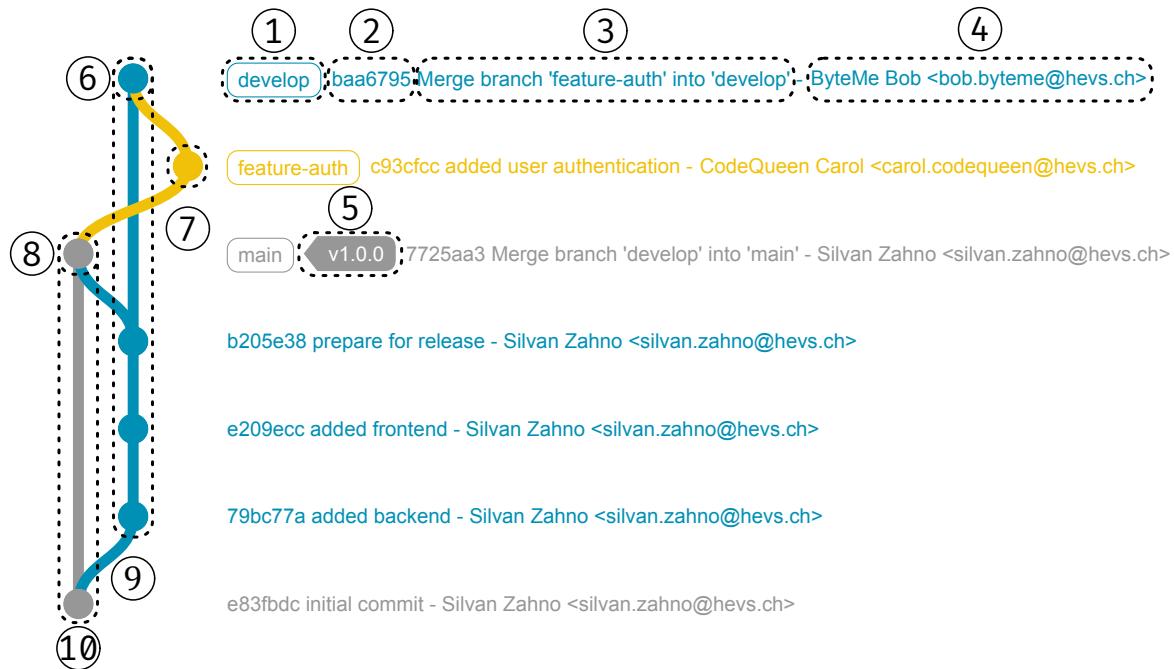


Abbildung 29 - Beispiel eines Gitgraphen

### Aufgabe 7

1. Benennen Sie alle Elemente die im Bild Abbildung 29 zu sehen sind (Punkte 1-10)
2. Committe deine Änderungen
3. Pushe dein lokales Repository in dein Cloud GitHub-Repository
4. Tag und pushe das Tag **chapter4** gemäss Abschnitt 6.10



Gratulation, die Arbeit an deinem ersten GitHub-Repository ist abgeschlossen.



# 9 | Gitflow

Verwenden Sie für diese Aufgabe die im Kurs vorgestellte Gitflow-Philosophie. Sie alle werden an dem folgenden Git-Repo zusammenarbeiten, als ob Sie ein Entwicklungsteam bilden würden:

<https://github.com/hei-synd-syd/syd-gitflow> [3]

Dies ist ein öffentliches Git-Repo, das auf Github gehostet wird.

## 9.1 Fork

Aus Sicherheitsgründen ist es Ihnen nicht gestattet, direkt an diesem Repository zu arbeiten. Sie müssen Ihre eigene Kopie (**fork**) erstellen, um Änderungen vornehmen zu können. Bitte erstellen Sie daher in Ihrem GitHub-Konto einen **fork** dieses Repositoriums. Verwenden Sie dazu die Schaltfläche **fork** in der Weboberfläche von Github.

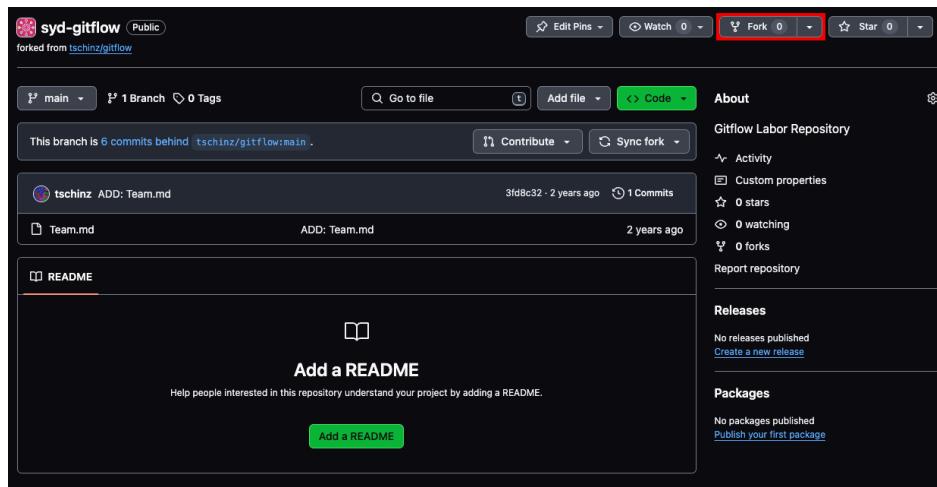


Abbildung 30 - Fork Knopf für ein GitHub Repository

Klonen Sie dann dieses geforkte Repo. Die URL Ihres neuen Repos wird dann wie folgt aussehen:

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

## 9.2 Parallele Zusammenarbeit



Bearbeiten Sie in einem lokalen Feature-Zweig die Datei **Team.md**. Ersetzen Sie Ihre gegebene Nummer durch Ihren Vornamen und Namen. **Committen** und **Pushen** Sie Ihren Branch in Ihr geforktes Repository auf GitHub.



Sie werden alle die gleiche Datei bearbeiten. Um Konflikte zu vermeiden, bearbeiten Sie nur die Zeile, die für Sie relevant ist.

Fragen Sie den Professor oder den Assistenten nach Ihrer Zeilennummer!



### 9.3 Pull Request

Jetzt, da Ihre Änderungen bereit sind, ist es noch notwendig, eine Zusammenführung im ursprünglichen Repository durchzuführen.

Da Sie keine Berechtigungen haben, ist es möglich, eine Zusammenführungsanfrage - **pull request** - im ursprünglichen Repository zu erstellen.

So kann der Administrator die Änderungen akzeptieren, Sie bitten, Elemente zu korrigieren, bevor Sie die Zusammenführung akzeptieren, Probleme diskutieren usw.

Für das verwenden Sie die Weboberfläche von GitHub. **Contribute** ⇒ **Open pull request**.

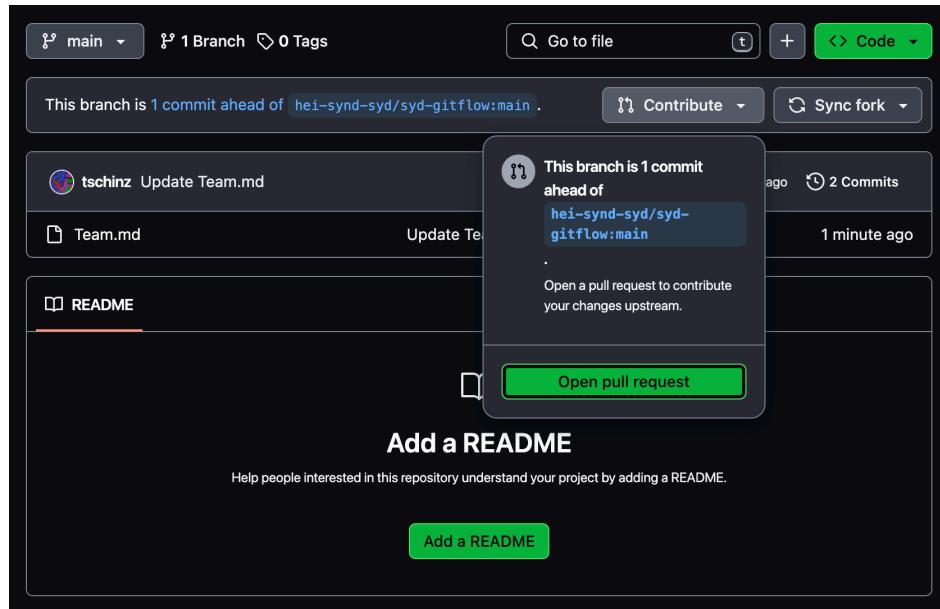


Abbildung 31 - Pull Request auf Github erstellen

Sie müssen Ihrer Pull-Request einen sprechenden Titel und eine kurze Beschreibung geben. Es ist möglich, die Beschreibung in Markdown zu schreiben - Abschnitt 3.

Die unvollständigen, schlecht beschriebenen, schlecht betitelten Pull-Requests ... werden abgelehnt.

Wenn die Pull-Requests bereit sind, werden die Zusammenführungen von dem Professor und Assistenten durchgeführt.



# 10 | Extras

Dieses optionelle Kapitel kann begonnen werden sofern die vorhergegangenen Aufgaben erledigt wurden. Es gibt 2 Aufgaben zu erledigen:

1. Eigenes Projekt auf Github legen und mit einem **README.md** versehen sowie einem CI/CD.
2. Folgen Sie dem Tutorial „Learn Git Branching“.

## 10.1 Eigenes Projekt

- Lege ein aktuelles Projekt an dem du arbeitest auf github ab.
- Erstelle **README.md** Datei für das Projekt mithilfe des [Markdown Syntaxes](#). Das **README.md** sollte folgendes beinhalten:
  - Titel
  - Bild
  - Beschreibung des Projektes
  - Erklärung wie man das Projekt ausführt / benutzt
  - Liste der Autoren
- Erstelle nun eine github action um das **README.md** in ein PDF zu verwandeln bei jedem Push. Suche hierzu eine passende [Github Action](#) und füge sie in dein Projekt ein.



Wenn Sie Hilfe bei der Erstellung der Github-Action benötigen. Schauen Sie sich den folgenden [Tipp](#) an.

Abbildung 32 - Marché de l'action Github



## 10.2 Git Branching lernen

Folgen Sie dem Tutorial auf <https://learngitbranching.js.org>.

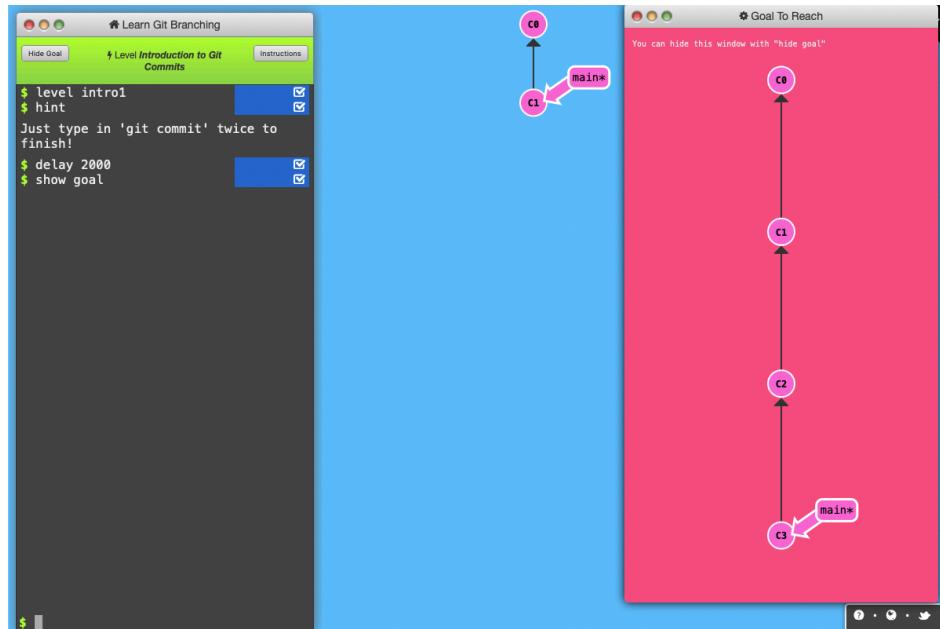


Abbildung 33 - Le site web de „apprendre branchement git“



# Literatur

- [1] T. Linus, „Git“. Zugegriffen: 25. April 2023. [Online]. Verfügbar unter: <https://git-scm.com/>
- [2] „Sublime Merge - Git Client from the Makers of Sublime Text“. Zugegriffen: 25. April 2023. [Online]. Verfügbar unter: <https://www.sublimemerge.com/>
- [3] tschinz, „Tschinz/Gitflow“. Zugegriffen: 25. April 2023. [Online]. Verfügbar unter: <https://github.com/tschinz/gitflow>
- [4] gitlab, „Git Cheatsheet“. 2023.
- [5] „GitHub Git Spickzettel“. Zugegriffen: 25. April 2023. [Online]. Verfügbar unter: <https://training.github.com/downloads/de/github-git-cheat-sheet/>



# 11 | Anhang

## A | GIT Befehle

[Github git cheatsheet](#) [4], [5]

### AA Änderungen überprüfen und eine Commit-Transaktion anfertigen

```
git status
```

Listet alle zum Commit bereiten neuen oder geänderten Dateien auf.

```
git diff
```

Zeigt noch nicht indizierte Dateiänderungen an.

```
git add [file]
```

Indiziert den derzeitigen Stand der Datei für die Versionierung.

```
git diff --staged
```

Zeigt die Unterschiede zwischen dem Index (“staging area”) und der aktuellen Dateiversion.

```
git reset [file]
```

Nimmt die Datei vom Index, erhält jedoch ihren Inhalt.

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

Nimmt alle derzeit indizierten Dateien permanent in die Versionshistorie auf.

### AB Änderungen synchronisieren

Registrieren eines externen Repositories (URL) und Tauschen der Repository-Historie.

```
git fetch [remote]
```

Lädt die gesamte Historie eines externen Repositories herunter.

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

Integriert den externen Branch in den aktuell lokal ausgecheckten Branch.



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

Pusht alle Commits auf dem lokalen Branch zu GitHub.

```
git pull
```

Pullt die Historie vom externen Repository und integriert die Änderungen.



# B | Meistgebrauchten Git Befehle

## 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