



Einführung Git - Teil A & B



Contents

1 Ziel	1
2 Installation	3
3 Markdown	6
4 Schlusswort	9
5 Ziele	10
6 Tools	10
7 Basis Operationen	11
8 Branch und Merge	19
9 Gitgraph	22
10 Gitflow	23
11 Extras	25
A GIT Befehle	27
B Meistgebrauchten Git Befehle	28
Bibliographie	29

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](#) kennenlernen [1], insbesondere die Tools [Git-Komandozeile](#) und [Sublime Merge](#), die auf Ihrem Rechner installiert und konfiguriert werden müssen (siehe Abschnitt 2). Des Weiteren werden Konten auf den Plattformen [Github](#) und [Hevs Gitlab](#) [2] erstellt.

Schlussendlich lernen wir die Grundlagen von Markdown in Abschnitt 3, um einfach Textdateien 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 sowie SublimeMerge. Sie können selber auswählen ob sie die Kommandozeile oder das GUI benutzen möchten während des Labors. Es sollten aber beide Tools installiert und konfiguriert werden.

2.1 git

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.

2.1.1 Kommandozeile

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

```

zas - zas@zac ~ ~ - zsh - 99x52
Last login: Tue Mar  8 09:26:26 on ttys004
[zas@zac ~] (base)
[ - $ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [-no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [-n <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 file contents 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)
bisect  Use binary search to find the commit that introduced a bug
diff    Show changes between commits, commit and working tree, etc
grep    Print lines matching a pattern
log    Show commit logs
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
reset   Reset current 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 from and integrate with 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.

[zas@zac ~] (base)
[ - $ ]

```

Abbildung 1: git Terminal

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



`git --help`



2.1.2 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
```

2.2 Sublime Merge

Besuchen Sie die Webseite <https://www.sublimemerge.com> und laden sowie installieren Sie das Tool Sublime Merge herunter [3].

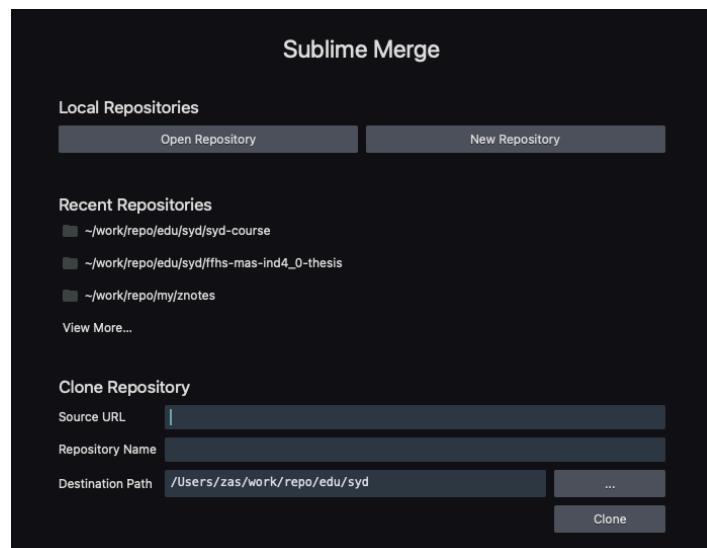


Abbildung 2: Sublime Merge GUI



2.3 Online-Konten

2.3.1 Gitlab

Besuchen Sie die Webseite <https://gitlab.hevs.ch> und loggen Sie sich mit Ihrem Schulkonto ein (SwitchEDU-ID).

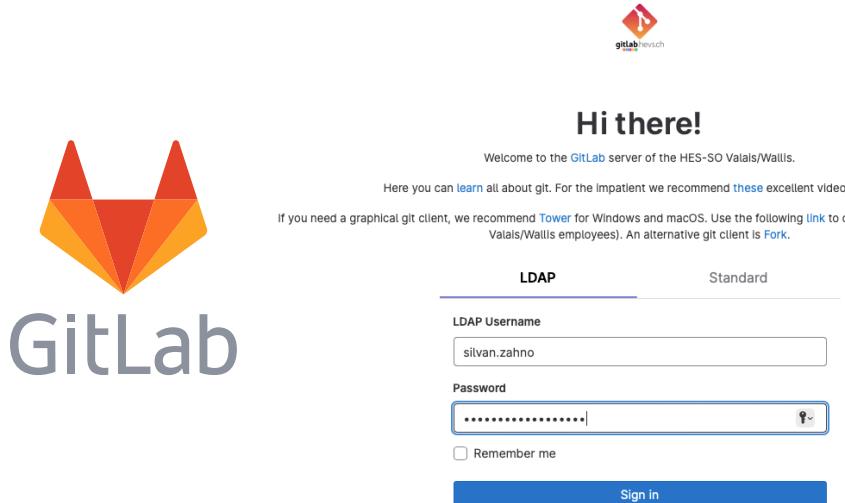


Abbildung 3: Gitlab Anmeldung

2.3.2 Github

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

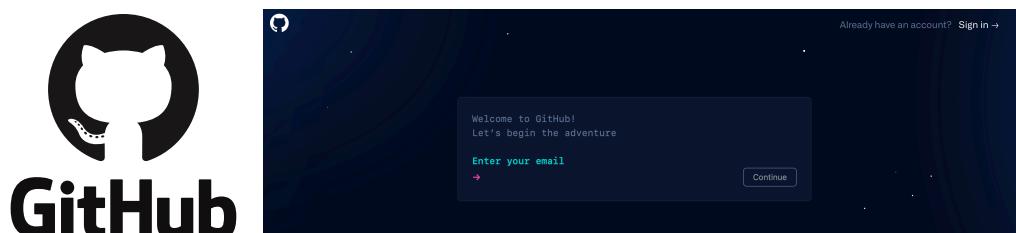


Abbildung 4: GitHub Anmeldung

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 ⇒ Ansicht ⇒ Anzeigen ⇒ Aktivieren Sie „**Dateinamenerweiterungen**“ und „**Versteckte Elemente**“

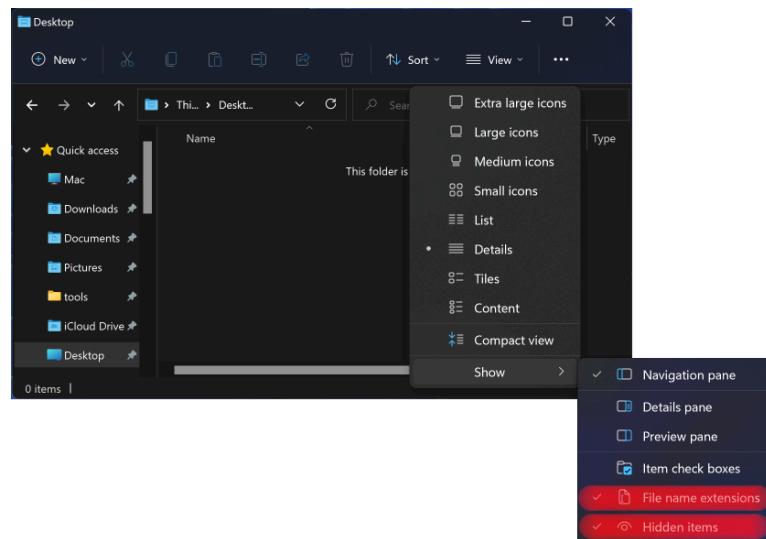


Abbildung 5: Windows Datei Explorer Konfiguration

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.

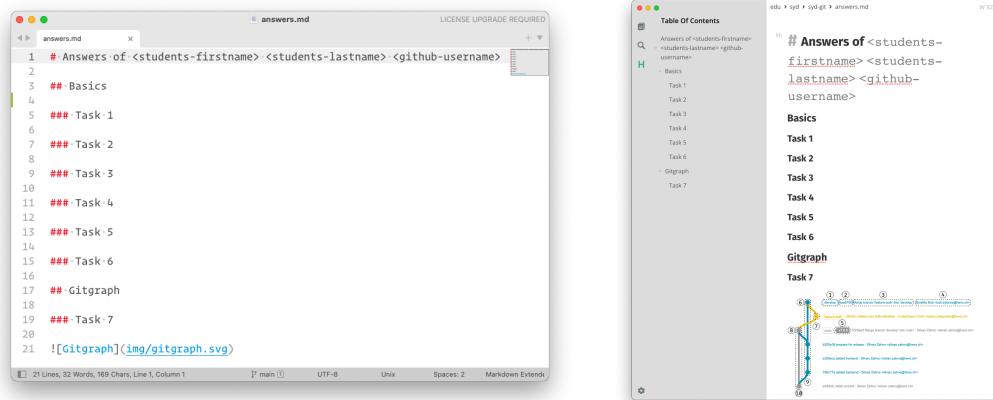


Abbildung 6: Links: Einfacher Texteditor (Sublime Text), Rechts: Marktext



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 [hier](#).



```

# 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 |

...

```

sources > 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

Glückwunsch Sie haben nun alles benötigte installiert und konfiguriert um mit Git zu arbeiten.

Bereiten Sie sich auf das nächste Labor vor:

- Studieren Sie die Theorie
- Familiarisieren Sie sich mit den Git Kommandos
- Familiarisieren Sie sich mit dem Grafischen Tool SublimeMerge
- Üben Sie das schreiben von Dokumenten mit Markdown



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 7 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 8. Im Abschnitt 10 arbeiten wir alle gemeinsam an einem Repository. Schlussendlich gibt es einige optionale Arbeiten im Abschnitt 11.



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.

6 | Tools

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



Git CMD
App



Sublime Merge
App

Abbildung 7: Git CMD Kommandozeile Abbildung 8: Sublime Merge GUI



7 | Basis Operationen

7.1 Erstellen eines git Repositories

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

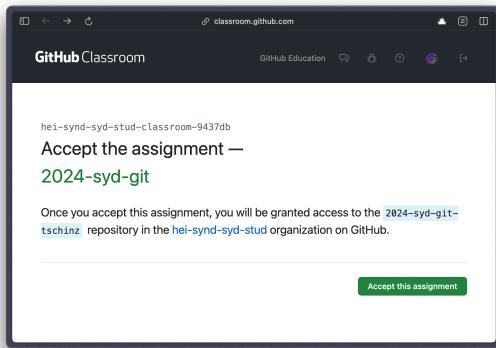


Abbildung 9: Einladungslink zum Forken

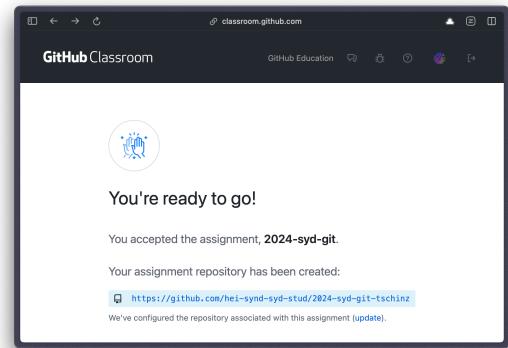


Abbildung 10: Link zum Klonen des Repository

7.2 Clone

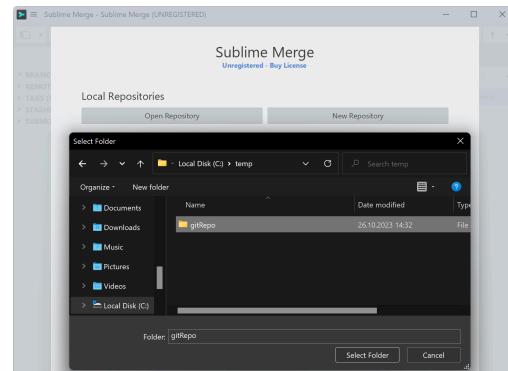
Nachdem Sie den Fork erstellt haben, erhalten Sie einen Link auf Ihr eigenes Repository (Abbildung 10). klonen Sie es auf Ihren lokalen Computer an einem von Ihnen bevorzugten Speicherort.

Commandline

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

GUI

CTRL+T ⇒ Paste Source URL ⇒ Select Folder



Aufgabe 0



Ändern Sie den Titel des Dokumentes `answers.md` und ersetzen Sie:

- <studentsfirstname> mit Ihrem Vornamen
- <studentslastname> mit Ihrem Nachnamen
- <githubusername> mit Ihrem GitHub Benutzernamen

**Aufgabe 1**

Was befindet sich im Verzeichnis, nachdem das Git-Repo geklont wurde? Für was sind die Verschiedenen Dateien und Ordner gut?

Schreiben Sie die Antworten in der Datei `answers.md` auf!

7.3 Status abfragen

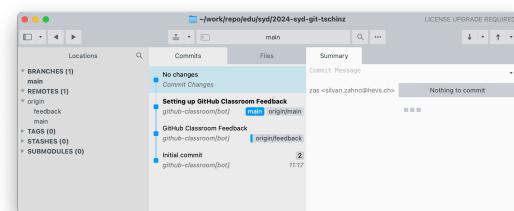
Erhalten Sie Informationen über Ihr Repo mit den folgenden Befehlen:

Commandline

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

GUI

See the status in the main window.



7.4 Datei hinzufügen

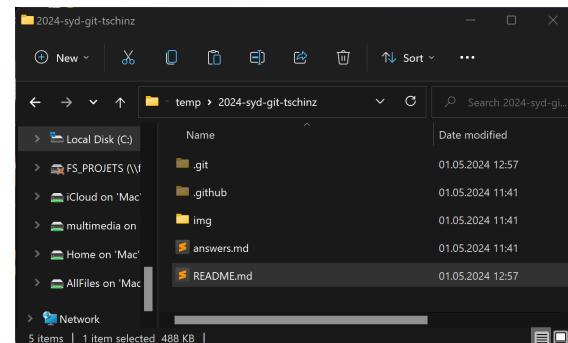
Erstellen Sie nun eine leere Datei mit dem Namen `README.md` im Hauptverzeichnis Ihres Repos.

Commandline

```
touch README.md
```

GUI

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



Verwenden Sie die vorherigen Befehle, um erneut die Informationen über Ihr Repo abzurufen. Was hat sich geändert?

**Aufgabe 2**

Was hat sich in `git status` und `git log -oneline` geändert? Und warum?
Schreiben Sie die Antwort auf!



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.

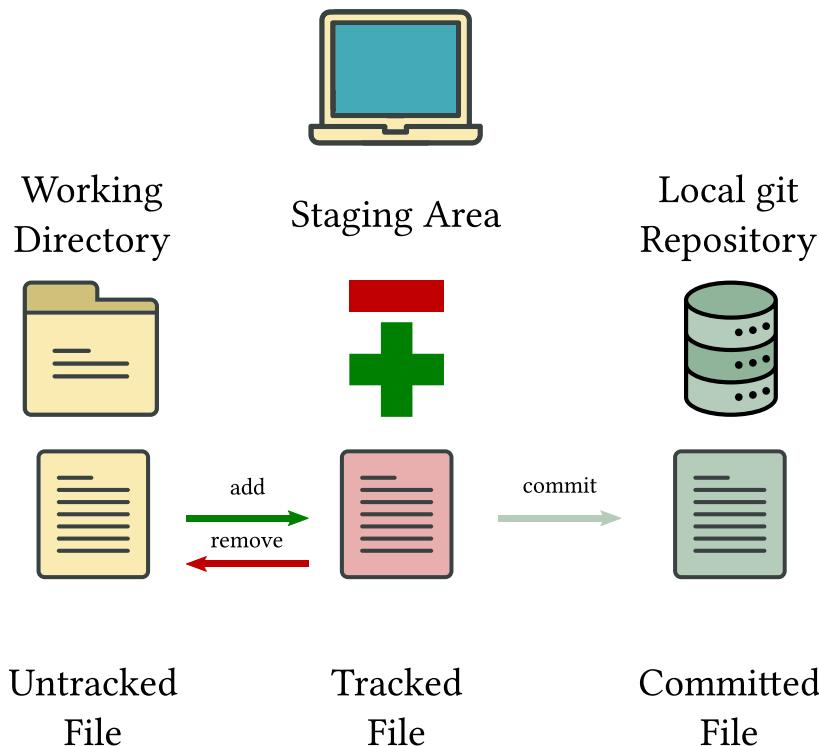


Abbildung 11: Arten von lokalen Git Operationen

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.

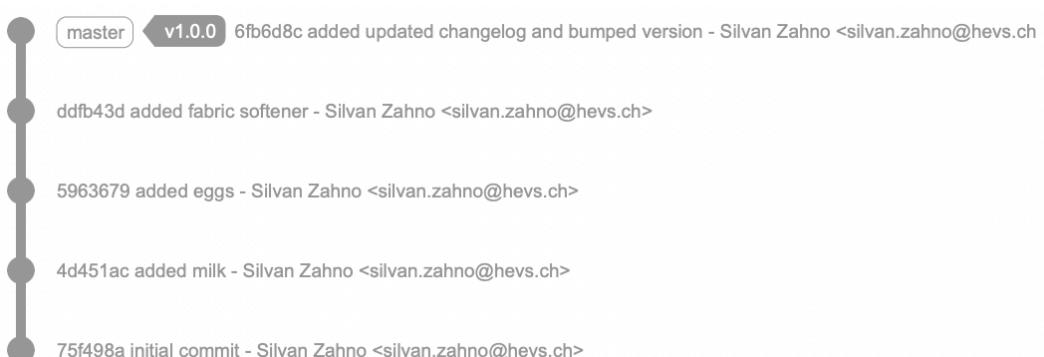


Abbildung 12: Fünf commits auf dem lokalen Repo, jeder commit besitzt seine eigene identifikation



7.5 Datei in Repo aufnehmen

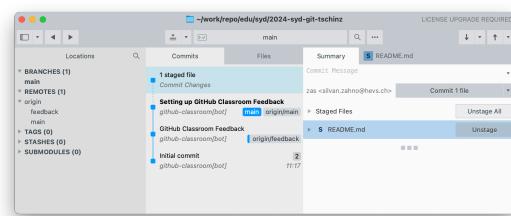
Fügen Sie die zuvor erstellte Datei `README.md` zur Stage hinzu, mit dem Befehl:

Commandline

```
git add README.md
```

GUI

Untracked Files \Rightarrow README.md \Rightarrow Stage



Aufgabe 3



Schauen Sie sich noch einmal die Infos von `git status` auf Ihrem Repo an, was haben Sie festgestellt?

Schreiben Sie die Antwort auf!

Bearbeiten Sie die Datei `README.md` mit einem Texteditor und fügen Sie den folgenden Text ein (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
```

```

Aufgabe 4



Schauen Sie sich noch einmal die Infos von `git status` auf Ihrem Repo an, was haben Sie festgestellt?

Schreiben Sie die Antwort auf!



7.6 Neue Änderungen hinzufügen

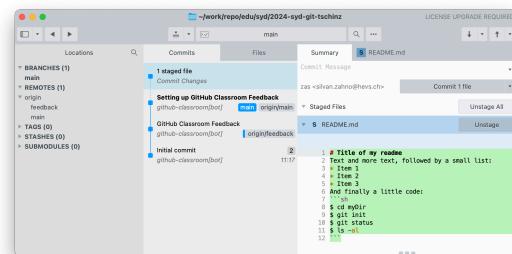
Fügen Sie die neueste Version der Datei README.md zum Stage hinzu.

Commandline

```
git add README.md
```

GUI

README.md \Rightarrow Stage



7.7 Commit ausführen

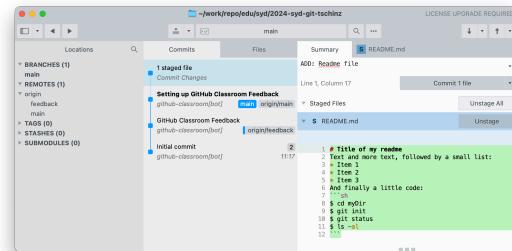
Führen Sie nun einen Commit mit folgendem Befehl durch:

Commandline

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

GUI

Commit Message \Rightarrow ADD: README file. \Rightarrow Commit 1 file



Die Option `-m` erlaubt es, die Nachricht des Commits direkt anzugeben. Diese Nachricht muss selbsterklärend sein. Sie entspricht der Beschreibung der Änderungen. Es ist möglich, einen Textblock z. B. über einen Texteditor einzufügen, ohne die Option `-m` zu verwenden.

Ihre Änderungen werden nun in Ihrem lokalen Git-Repo veröffentlicht. Bravo!



7.8 Mehr informationen

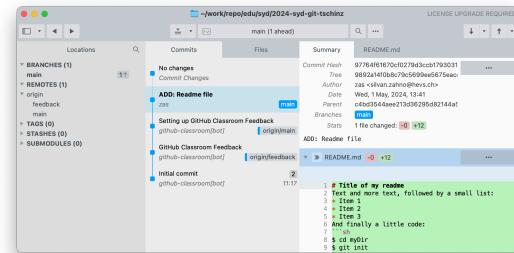
Welche Informationen erhalten Sie nun mit dem Befehl:

Commandline

GUI

Select First Commit ⇒ See all informations

```
git log --oneline
```



Aufgabe 5

Erklären Sie alle Informationen in der ersten Zeile deutlich.



- Was bedeutet die Zeichenkette am Anfang?
 - Was bedeuten `HEAD` und `main`?
 - Was steht hinter den Klammern?

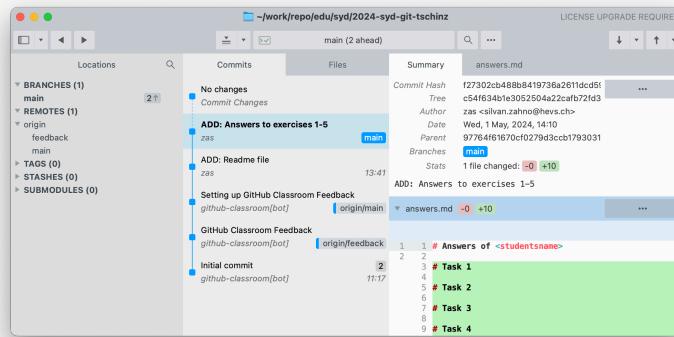
Schreiben Sie die Antwort auf!



7.9 Checkout commit

Damit Ihre Antworten nicht verloren gehen, legen Sie die Änderungen der Datei `answers.md` in einem commit ab.

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



Jeder Commit ist mit einem “Hash” oder einer “Prüfsumme” (vom Typ sha1) versehen wird. Die mit dem Befehl:

```
git log --oneline
```

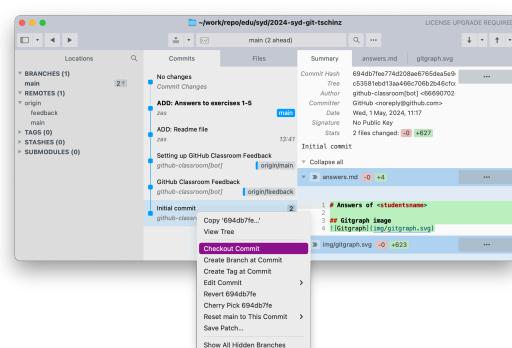
die angezeigten Hashes sind nur die ersten paar Zeichen dieser sogenannten “short hashes”. Führen Sie nun einen Checkout mit dem folgenden Befehl durch und verwenden Sie dabei den “short hash”, der dem ersten Commit entspricht.

Commandline

GUI

Select First Commit (Initial Commit) ⇒
Checkout commit

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



Schauen Sie sich nun den Inhalt des Ordners genau an.



7.10 Checkout master

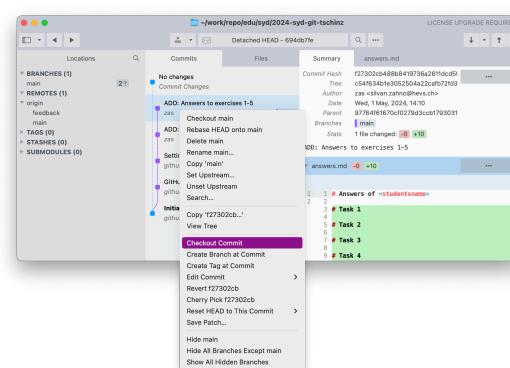
Führen Sie nun einen Checkout mit dem folgenden Befehl durch:

Commandline

```
git checkout main
```

GUI

Branches (1) \Rightarrow master \Rightarrow checkout master



Aufgabe 6

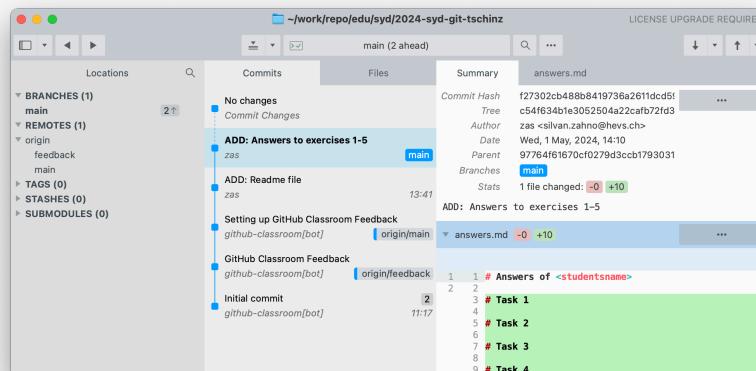


Was haben Sie bemerkt was ist mit den Dateien und Ordner im Projektordner passiert als Sie den „Initial Commit“ genommen hatten? Was ist passiert wenn die wieder auf den letzten Commit gesprungen sind?

Schreiben Sie die Antwort auf!

Damit Ihre Antworten nicht verloren gehen, legen Sie die Änderungen der Datei `answers.md` in einem commit ab.

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





8 | Branch und Merge

Bisher haben wir die grundlegenden Funktionen von git verwendet. Es gibt auch die Funktionen branch (Zweig) und merge (zusammenführen), die Git im Vergleich zu den früher existierenden Tools stark vereinfacht hat.

Für diese praktische Arbeit können Sie sich mit dem GUI Sublime Merge behelfen, der Ihnen eine grafische Darstellung und einen visuellen Verlauf der Commits in Ihrem Repo liefert.

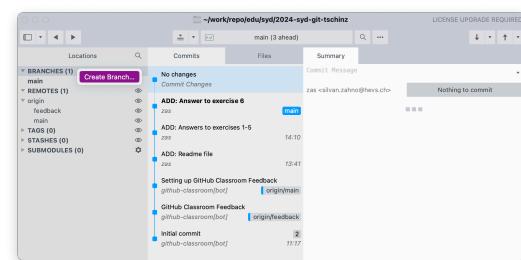
1. Erstellen Sie einen Entwicklungszweig dev01 in Ihrem lokalen Repo.

Commandline

```
git checkout -b dev01
```

GUI

Branches ⇒ Create Branch ⇒ dev01



2. Erstellen Sie einen Commit auf diesem Zweig:

- Um eine Datei `hello_world.py` zu erstellen und zu füllen.

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

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

3. Checkout des `main` Zweigs

```
git checkout main
```

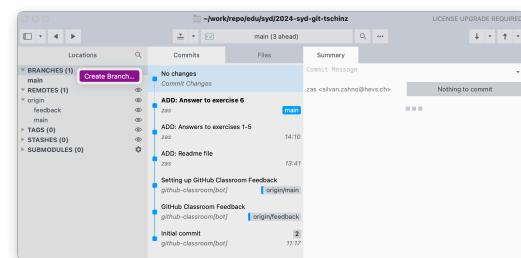
4. Ausgehend vom `main`-Zweig Erstellen Sie einen neuen Entwicklungszweig `dev02`.

Commandline

```
git checkout -b dev02
```

GUI

Branches ⇒ Create Branch ⇒ dev02



5. Erstellen Sie einen Commit auf diesem Zweig:



- Um eine Datei `hello_world.rs` zu erstellen und zu füllen.

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

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

6. Checkout des `main`-Zweigs.

```
git checkout main
```

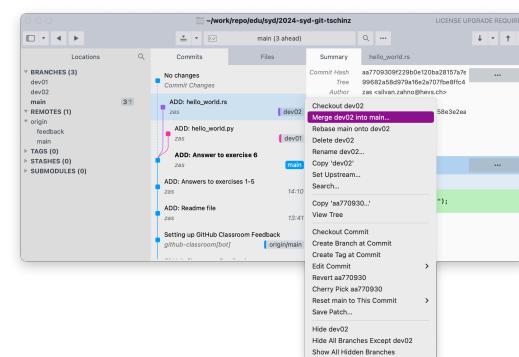
7. Merge den Zweig `dev02` in `main`.

Commandline

```
git merge dev02
```

GUI

Select Commit \Rightarrow Merge `dev02` into `main` ...



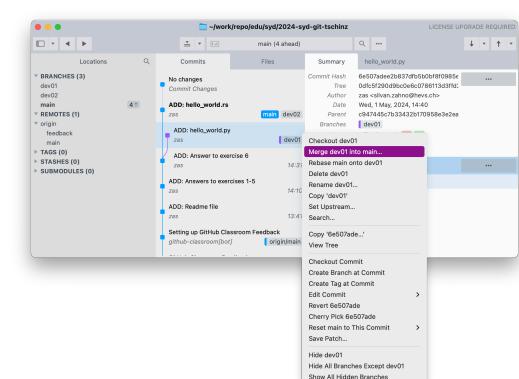
8. Merge den Zweig `dev01` in `main`.

Commandline

```
git merge dev01
```

GUI

Select Commit \Rightarrow Merge `dev01` into `main` ...



9. Pushe dein lokales Repository in dein Cloud GitHub-Repository.



Commandline

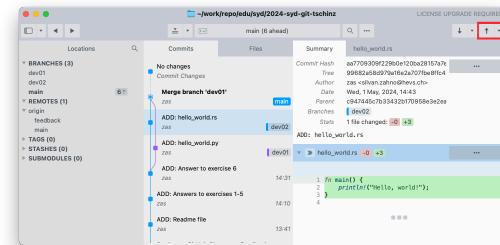
```
git push origin main
git push origin dev01
git push origin dev02
```

GUI

Checkout dev01 ⇒ Top Right Arrow ⇒ Push changes

Checkout dev02 ⇒ Top Right Arrow ⇒ Push changes

Checkout main ⇒ Top Right Arrow ⇒ Push changes



8.1.1 Endresultat

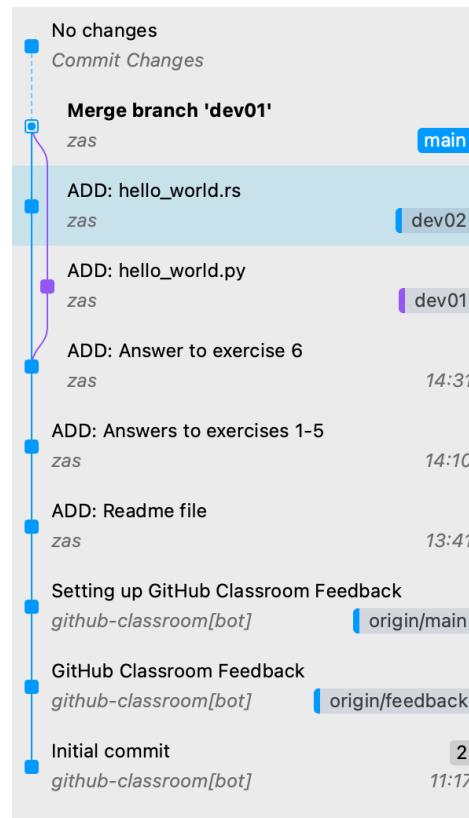


Abbildung 18: Status nach der Zusammenführung der Repository-Zweige dev01 und dev02



Der Status Ihres Repositories sollte ähnlich der Abbildung 18 entsprechen. Dies ist ein Teil der Bewertung.



9 | Gitgraph

In der Abbildung 19 ist ein Beispiel eines Git Repositories dargestellt. Benennen Sie alle Elemente die im auf dem Bild zu erkennen sind (Punkte 1- 10).

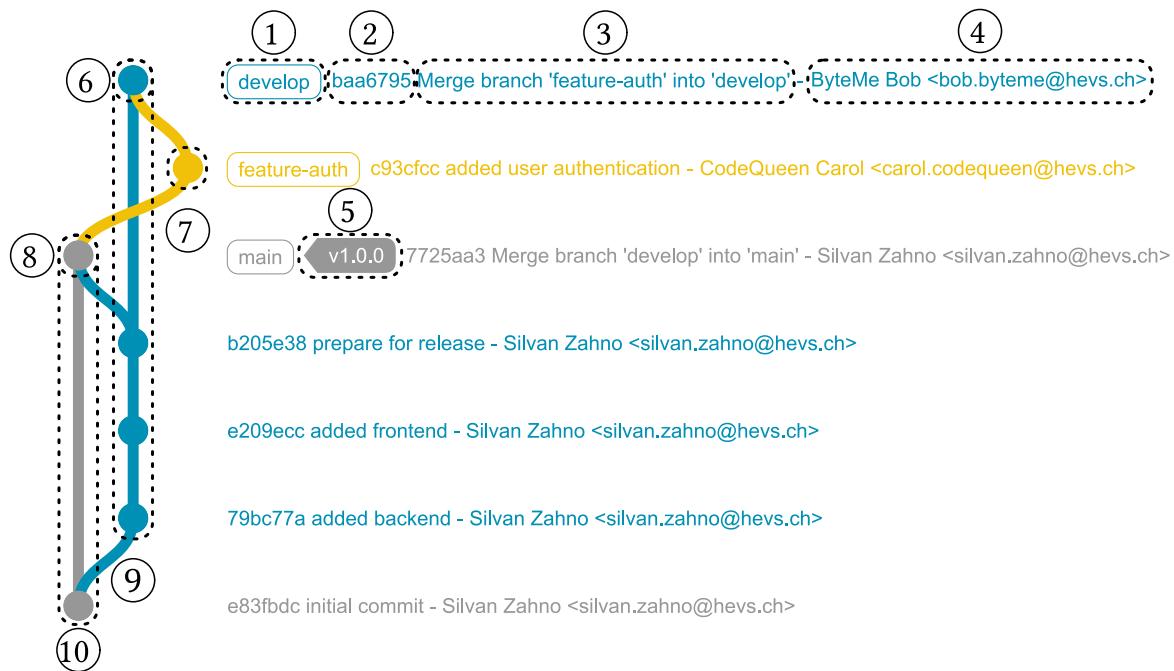


Abbildung 19: Beispiel eines Gitgraphen

Aufgabe 7



1. Benennen Sie alle Elemente die im Bild Abbildung 19 zu sehen sind (Punkte 1-10).
2. Pushe dein lokales Repository in dein Cloud GitHub-Repository.



10 | 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> [4]

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

10.1 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.

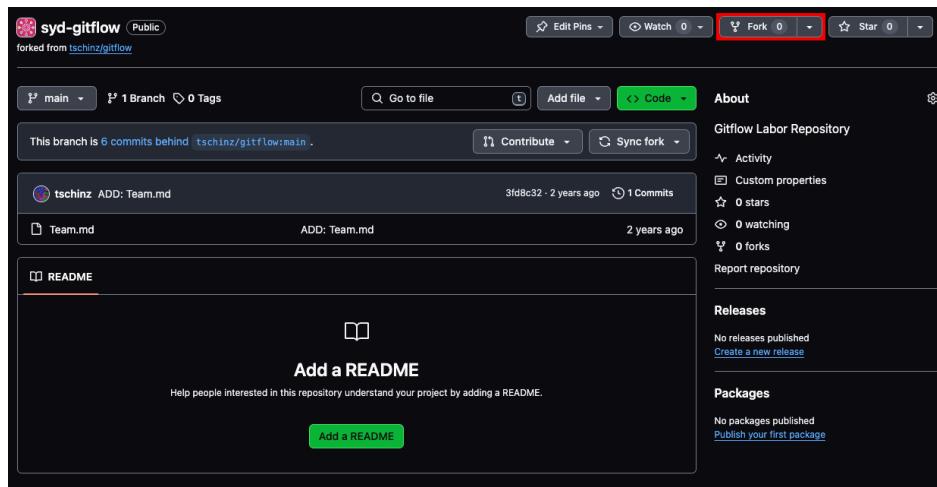


Abbildung 20: Fork Knopf für ein GitHub Repository

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
```

10.2 Parallele Zusammenarbeit

Bearbeiten Sie in einem lokalen Feature-Zweig die Datei `Team.md`. Ersetzen Sie Ihre gegebene Nummer durch Ihren Vornamen und Namen.



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

"Commiten" und "Pushen" Sie Ihren Zweig in das Fork-Repository auf GitHub.



10.3 Pull Request

Erstellen Sie einen “Pull Request” (Merge-Anfrage) auf GitHub. Verwenden Sie dazu die Schnittstelle der GitHub-Website.

Sobald alle Pull Requests fertig sind, werden die Merges in Absprache mit der gesamten Gruppe (und den Lehrern) durchgeführt.

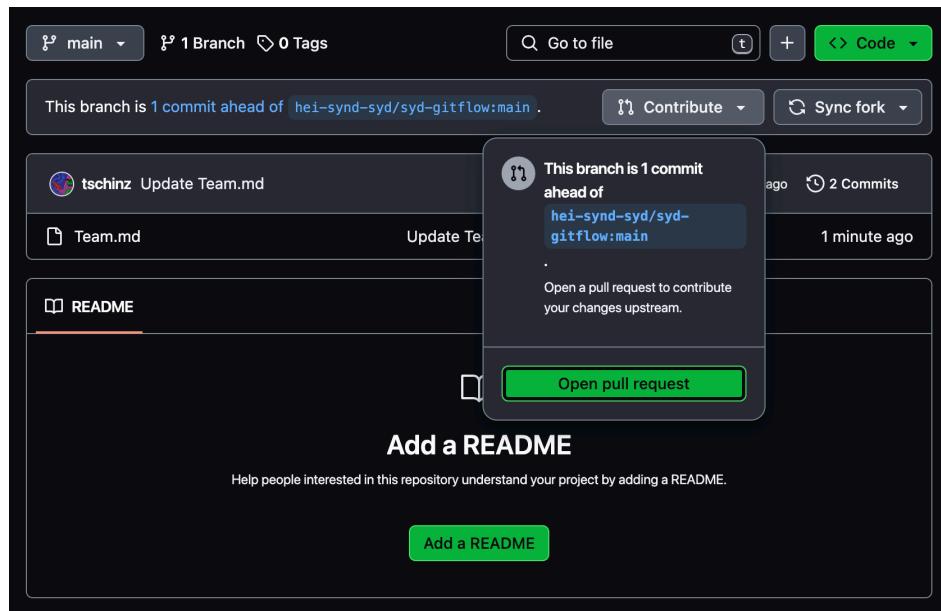


Abbildung 21: Pull Request auf Github erstellen



11 | 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“.

11.1 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.

The screenshot shows the GitHub Marketplace interface. The left sidebar has a navigation menu with 'Marketplace' selected. Under 'Actions', there are categories like 'API management', 'Chat', 'Code quality', 'Code review', 'Continuous integration', 'Dependency management', 'Deployment', 'IDEs', 'Learning', and 'Localization'. The main area is titled 'Actions' with the sub-instruction 'An entirely new way to automate your development workflow.' Below this, it says '20351 results filtered by Actions'. There are four cards displayed:

- Close Stale Issues**: By actions (verified), Close issues and pull requests with no recent activity, 1.1k stars.
- Upload a Build Artifact**: By actions (verified), Upload a build artifact that can be used by subsequent workflow steps, 2.5k stars.
- Download a Build Artifact**: By actions (verified), Download a build artifact that was previously uploaded in the workflow by the upload-artifact action, 1.1k stars.
- Setup Java JDK**: By actions (verified), Set up a specific version of the Java JDK and add the command-line tools to the PATH, 1.3k stars.

Abbildung 22: Marché de l'action Github



11.2 Git Branching lernen

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

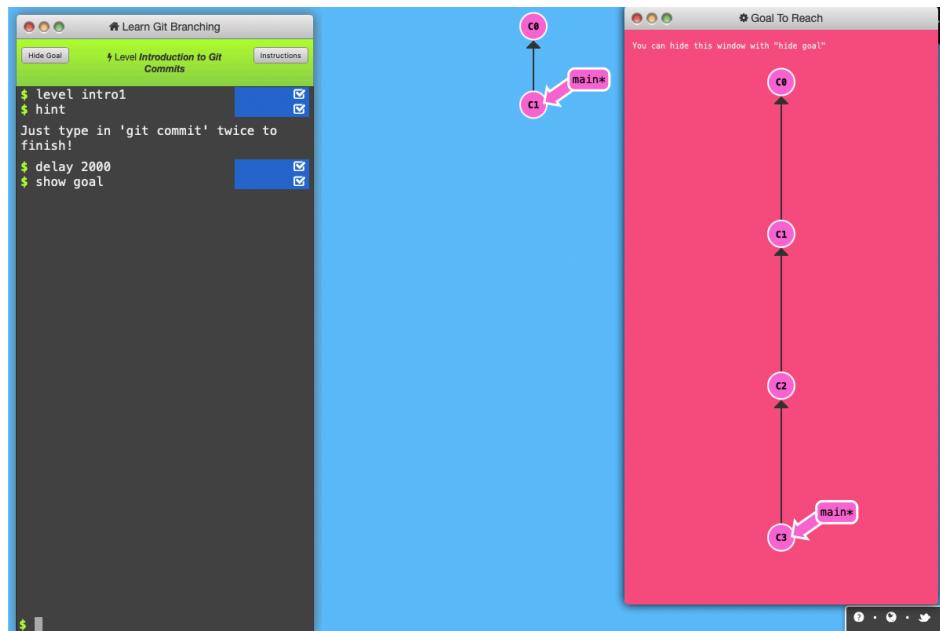


Abbildung 23: Le site web de „apprendre branchement git“



A | GIT Befehle

[Github git cheatsheet](#) [5], [6]

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



Bibliographie

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