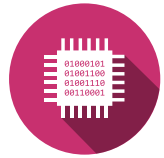


Einführung in die Git-Dateiversionierung



Inhalt

1	Ziel	2
2	Installation	3
2.1	git	3
2.2	Sublime Merge	4
2.3	Online-Konten	4
2.4	Windows-Konfiguration	5
3	Basis Operationen	7
3.1	Erstellen eines git Repositories	7
3.2	Initialisieren	7
3.3	Status abfragen	8
3.4	Datei hinzufügen	8
3.5	Datei in Repo aufnehmen	9
3.6	Neue Änderungen hinzufügen	10
3.7	Commit ausführen	11
3.8	Mehr Informationen	11
3.9	Weitere commits	12
3.10	Checkout commit	13
3.11	Checkout master	14
4	Branch und Merge	15
5	Gitflow	17
5.1	Fork	17
5.2	Parallele Zusammenarbeit	17
5.3	Pull Request	18
6	Extras	19
6.1	Eigenes Projekt	19
6.2	Git Branching lernen	20



1 | Ziel

In diesem Labor lernen wir die Grundprinzipien von der Versionskontrolle **git** kennen [1].

Zu Beginn werden wir Git auf Ihrem Rechner installieren und konfigurieren siehe Abschnitt 2. Sowie Konten auf [Github](#) [2] sowie [Hevs Gitlab](#) [3] erstellen.

Im Abschnitt 3 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 4.

Im Abschnitt 5 arbeiten wir alle gemeinsam nach dem Gitflow Prinzip.

Schlussendlich gibt es einige optionale Arbeiten im Abschnitt 6.



Die Antworten auf die Fragen sollten in einer Markdown-Datei niedergeschrieben werden. Die Datei sollte **answers.md** heissen.



2 Installation

Der erste Schritt ist die Installation von Git sowie Sublimemerge.

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". Dies ist ein Unix/Linux-ähnlicher Befehlseditor, der es ermöglicht, Git-Befehle im Konsolenmodus auszuführen. Diese Schnittstelle werden wir in diesem Labor verwenden.

```

Last login: Tue Mar  8 09:26:26 on ttys004
(zas@zas)~ (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>] [--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 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@zas)~ (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.

```
1 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.

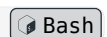
Wir werden nun die Minimalkonfiguration durchführen. Verwende die folgenden Befehle, um deine Identität in Git global auf dem System einzustellen. Verwende deinen Namen und deine E-Mail-Adresse. Diese Informationen sind öffentlich sichtbar, um deine Arbeit (deine Commits) zu identifizieren.

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



Sie können die Konfiguration mit dem folgenden Befehl überprüfen:

```
1 git config --list
```



Sie können auch eine bestimmte Einstellung überprüfen:

```
1 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 [4].

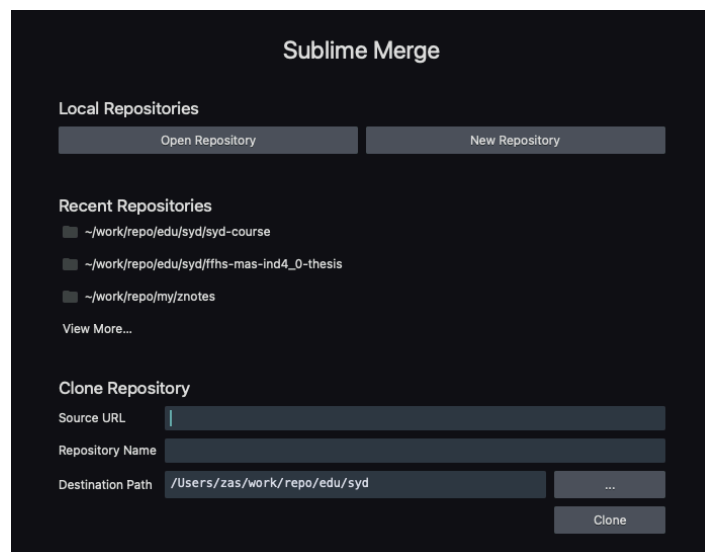


Abbildung 2 - Sublime Merge GUI

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).

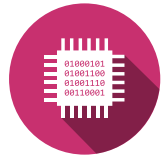


Tabelle 1 - Gitlab Anmeldung

2.3.2 Github

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

Tabelle 2 - 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“**

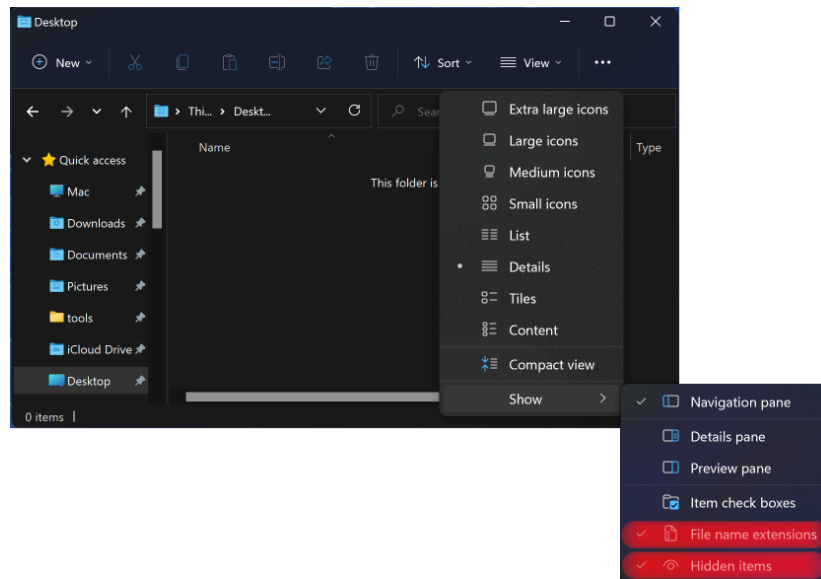
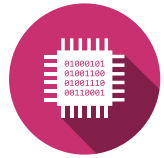
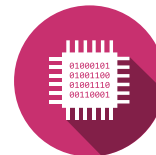


Abbildung 3 - Windows Datei Explorer Konfiguration



3 | Basis Operationen

3.1 Erstellen eines git Repositories

Erstellen Sie mithilfe der Git Bash Konsole ein leeres Verzeichnis auf Ihrem Computer, z.B. **C:\temp\gitRepo** oder **~/tmp/gitRepo**. Dieses Verzeichnis wird Ihr Git-Repository sein.

Du kannst die Unix/Linux-Befehle **ls** (list files and directories), **cd** (change directory), **pwd** (print current working directory) und **mkdir** (make directory) verwenden, um dieses Verzeichnis zu erstellen.

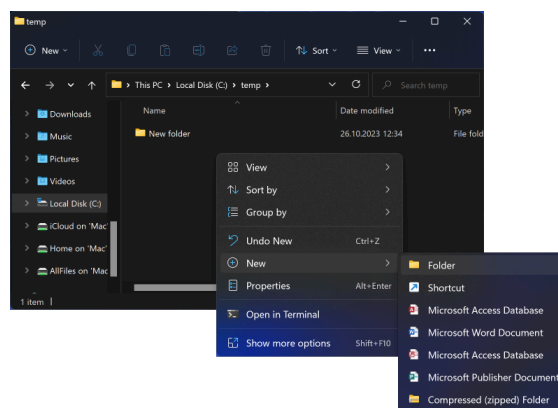
Commandline

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



GUI

C:\temp ⇒ New ⇒ Folder ⇒ gitRepo



3.2 Initialisieren

Sobald du in diesem Verzeichnis bist, initialisiere es als Git Repo mit dem Befehl:

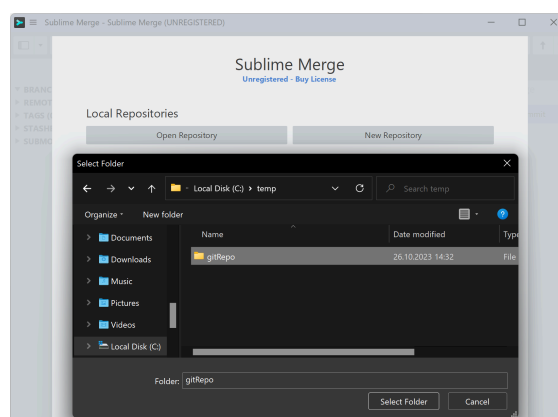
Commandline

```
1 git init
```

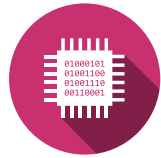


GUI

CTRL+T ⇒ New Repository ⇒ Select Folder



Was hat sich in dem Verzeichnis geändert, nachdem du es als Git-Repo initialisiert hast?



Schreiben Sie die Antwort auf!

3.3 Status abfragen

Erhalten Sie Informationen über Ihr Repo mit den folgenden Befehlen:

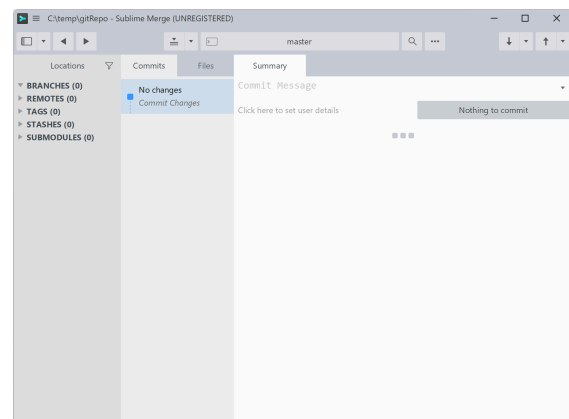
Commandline

```
1 git status
2 git log --oneline
```



GUI

See the status in the main window.



3.4 Datei hinzufügen

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

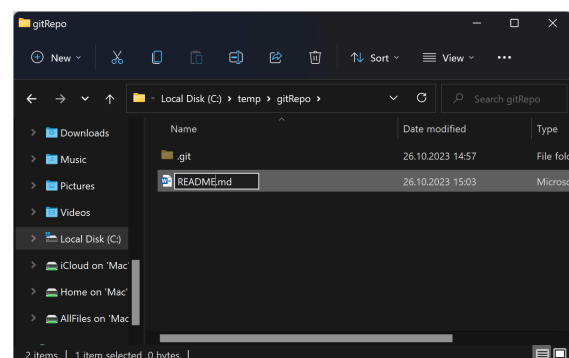
Commandline

```
1 touch README.md
```



GUI

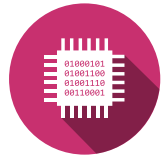
C:\temp ⇒ New ⇒ Any File ⇒ README.md



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



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.

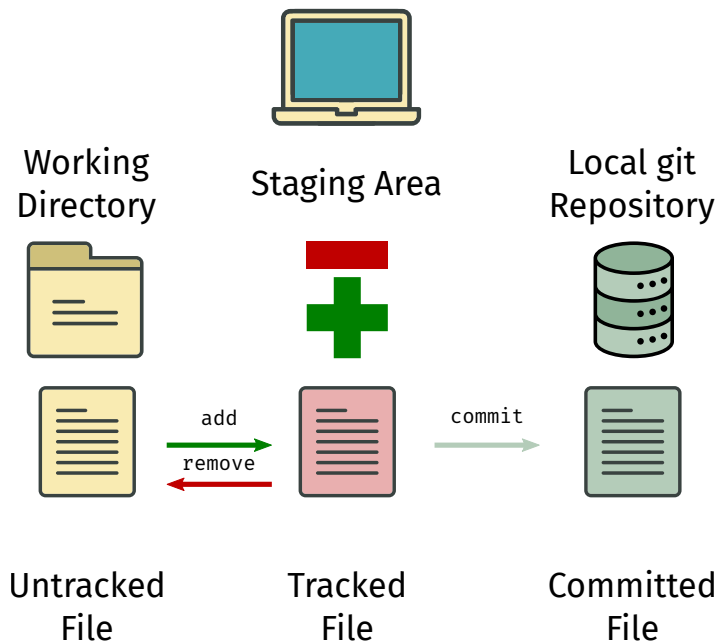


Abbildung 4 - Arten von lokalen Git Operationen

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.

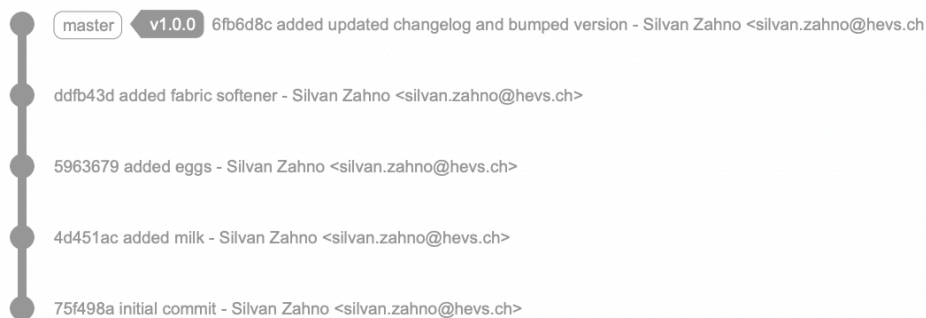


Abbildung 5 - Fünf commits auf dem lokalen Repo, jeder commit besitzt seine eigene identifikation

3.5 Datei in Repo aufnehmen

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

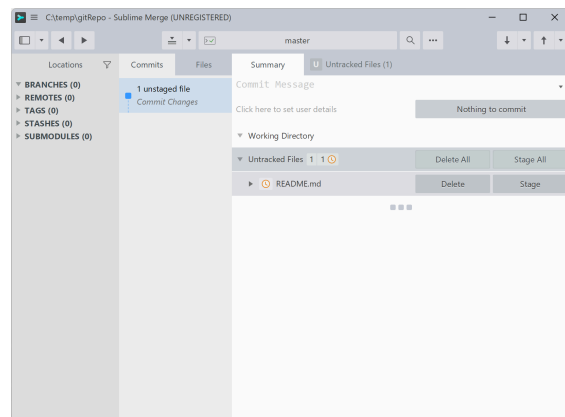
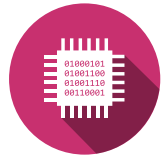
Commandline

GUI

Untracked Files ⇒ README.md ⇒ Stage

```
1 git add README.md
```

Bash



Rufen Sie erneut die Informationen über Ihr Repo ab, was stellen Sie fest?



Schreiben Sie die Antwort auf!

Bearbeiten Sie die Datei **README.md** mit einem texteditor und fügen Sie den folgenden Text ein (Markdown-Syntax):

```
1
2 # Title of my readme
3 Text and more text, followed by a small list :
4 * Item 1
5 * Item 2
6 * Item 3
7
8 And finally a little code:
9 ```sh
10 $ cd myDir
11 $ git init
12 $ git status
13 $ ls -al
14 ```
```

Markdown

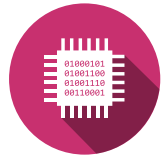
Erhalte erneut die Informationen über dein Repo, was stellst du fest?



Schreiben Sie die Antwort auf!

3.6 Neue Änderungen hinzufügen

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



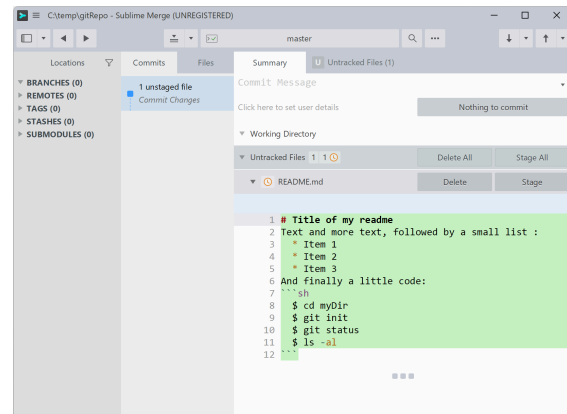
Commandline

```
1 git add README.md
```



GUI

Untracked Files ⇒ README.md ⇒ Stage



3.7 Commit ausführen

Führen Sie nun einen Commit mit folgendem Befehl durch:

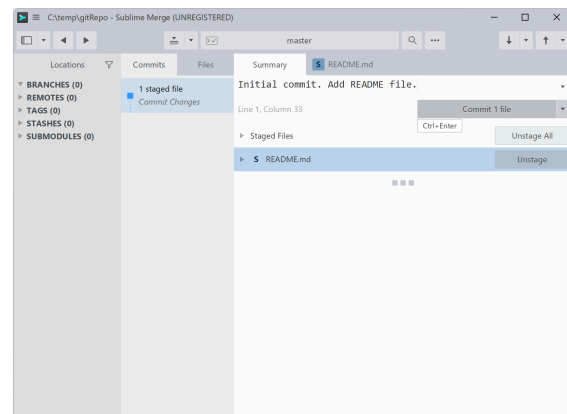
Commandline

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



GUI

Commit Message ⇒ Initial commit. Add README file. ⇒ Commit 1 file

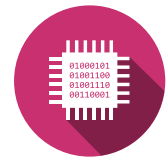


Die Option **-m** erlaubt es, die Nachricht des Commits direkt anzugeben. Diese Nachricht muss selbsterklärend sein. Sie entspricht der Beschreibung der Änderungen. Es ist möglich, einen Textblock z. B. über einen Texteditor einzufügen, ohne die Option **-m** zu verwenden.

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

3.8 Mehr Informationen

Welche Informationen erhalten Sie nun mit dem Befehl:



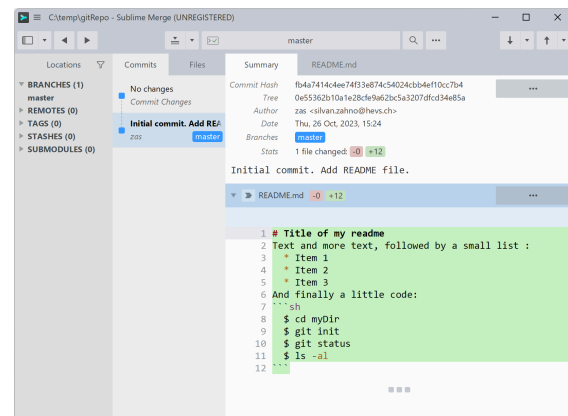
Commandline

```
1 git log --oneline
```



GUI

Select First Commit ⇒ See all informations



Erläutern Sie alle Informationen in dieser Zeile deutlich.



Schreiben Sie die Antwort auf!

3.9 Weitere commits

Führen Sie einen weiteren Commit durch, um eine neue (leere) **hello_world.py** Datei in Ihr Repo aufzunehmen.

Welche neuen Informationen liefert Ihnen nun der Befehl:

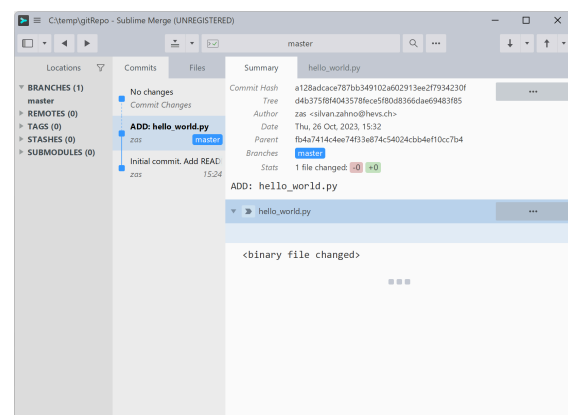
Commandline

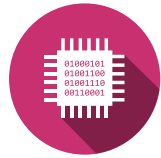
```
1 git log
```



GUI

Select Second Commit ⇒ See all informations





Schreiben Sie die Antwort auf!

Beachten Sie, dass jeder Commit mit einem “Hash” oder einer “Prüfsumme” (vom Typ sha1) versehen wird. Die mit dem Befehl:

```
1 git log --oneline
```

Bash

angezeigten Hashes sind nur die ersten paar Zeichen dieser sogenannten “short hashes”.

3.10 Checkout commit

Führen Sie nun einen Checkout mit dem folgenden Befehl durch und verwenden Sie dabei den “short hash”, der Ihrem ersten Commit entspricht.

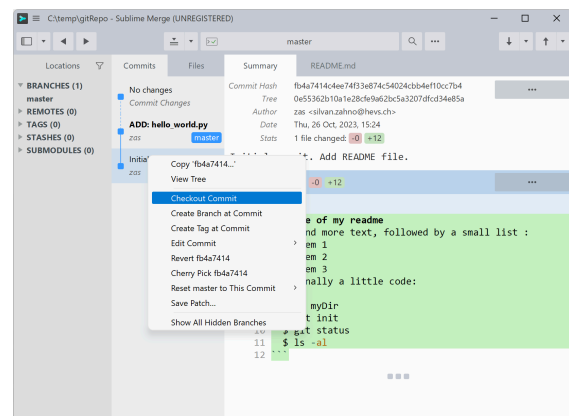
Commandline

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

Bash

GUI

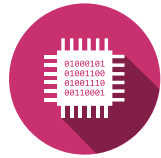
Select First Commit ⇒ Checkout commit



Was stellen Sie nun fest, wenn Sie den Inhalt des Working Directory aufrufen?



Schreiben Sie die Antwort auf!



3.11 Checkout master

Führen Sie nun einen Checkout mit dem folgenden Befehl durch:

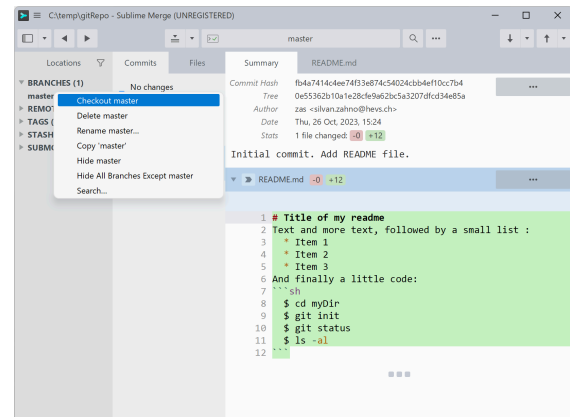
Commandline

```
1 git checkout master
```



GUI

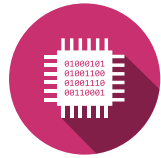
Branches (1) ⇒ master ⇒ checkout master



Was stellen Sie nun fest, wenn Sie den Inhalt des working directory betrachten?



Schreiben Sie die Antwort auf!



4 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 ein Github Repo und fügen Sie diesen als Remote hinzu

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Repository template

No template

Start your repository with a template repository's contents.

Owner * **Repository name ***

tschinz / gitRepo

gitRepo is available.

Great repository names are short and memorable. Need inspiration? How about [stunning-octo-bassoon](#) ?

Description (optional)

test repo for course SyD

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

☐ You are creating a public repository in your personal account.

Create repository

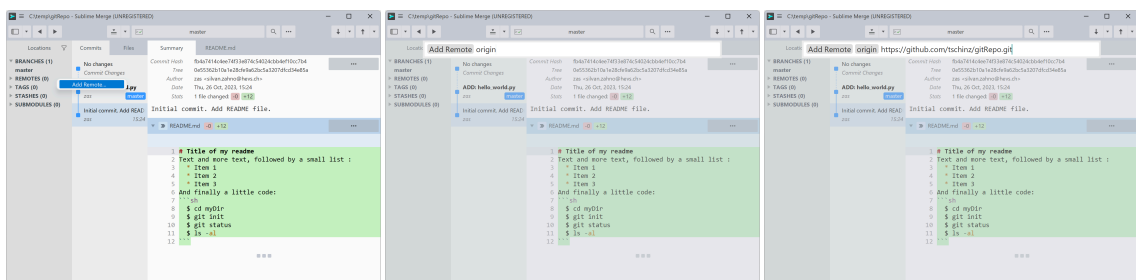
Commandline

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

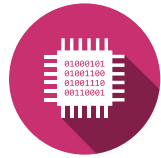


GUI

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



2. Erstellen Sie in Ihrem lokalen Repo einen Entwicklungszweig **dev01**.



3. Erstellen Sie auf diesem Zweig zwei Commits:
 - Einen, um eine Datei **hello_world.py** zu erstellen (dieser Commit existiert bereits).
 - Eines, um diese Datei **hello_world.py** zu füllen.
- ```
1 print("Hello, world!")
```

Python
4. Checkout des **master** Zweiges
  5. Vom master Branch aus erstellen Sie einen neuen Entwicklungszweig **dev02**.
  6. Bearbeiten Sie die Datei **README.md** und erstellen Sie einen commit auf dem Zweig **dev02**.
  7. Mergen Sie den Zweig **dev02** in **master**.
    - Checken Sie den **master**-Zweig aus
    - Führen Sie das merge „merge **dev02** into **master**“ durch.
  8. Zusammenführen des Zweigs **dev01** in **master**.
    - Checken Sie den **master**-Zweig aus
    - Führen Sie das merge „merge **dev01** into **master**“ durch.
  9. Pushen Sie Ihr lokales Repository auf Ihr GitHub-Remote.  
**git push origin master.**

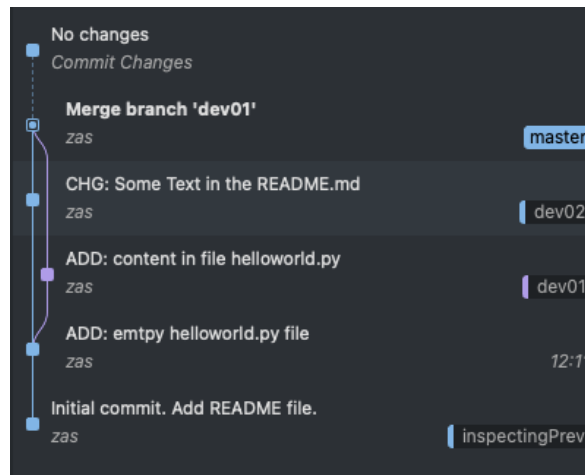


Abbildung 7 - Status nach der Zusammenführung der Repository-Zweige **dev01** und **dev02**



## 5 | 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/tschinz/gitflow> [5]

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

### 5.1 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.

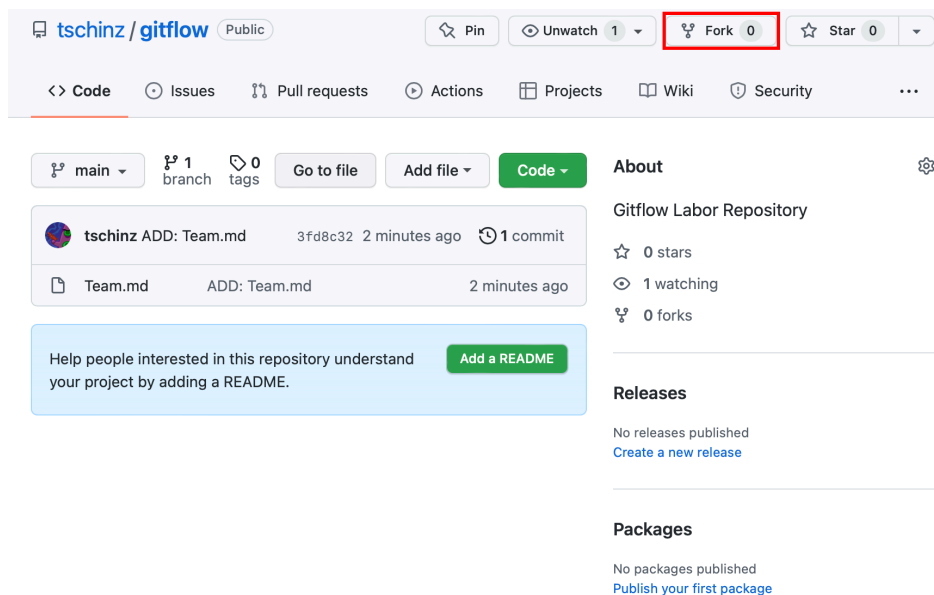
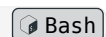


Abbildung 8 - Fork Knopf für ein GitHub Repository

Klonen Sie anschliessend unser gemeinsames Repository. Die URL Ihres Repository wird ähnlich aussehen wie:

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



### 5.2 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.



### 5.3 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.



## 6 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“.

### 6.1 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.

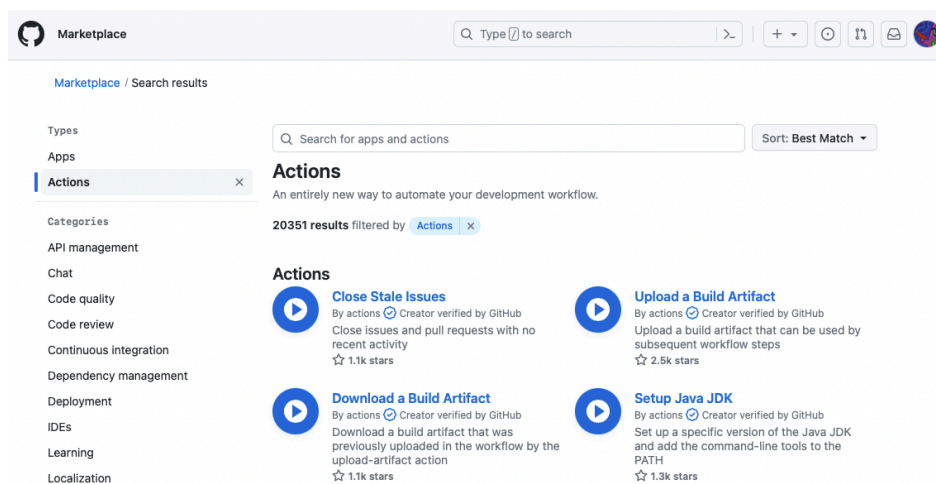


Abbildung 9 - Marché de l'action Github



## 6.2 Git Branching lernen

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

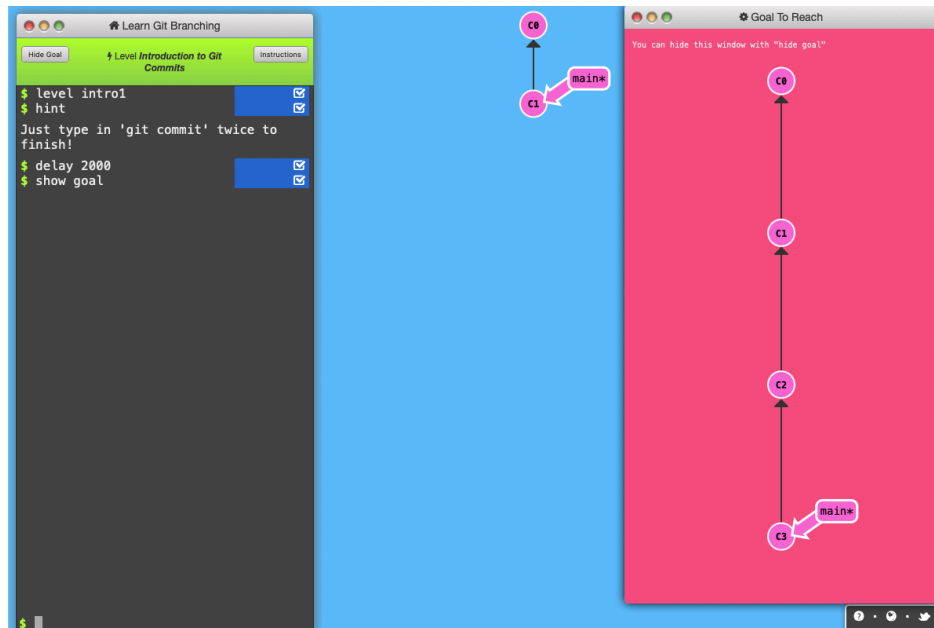


Abbildung 10 - Le site web de „apprendre branchement git“



# A | GIT Befehle

[Github git cheatsheet](#) [6], [7]


## AA Änderungen überprüfen und eine Commit-Transaktion anfertigen

```
1 git status
```

 Bash


Listet alle zum Commit bereiten neuen oder geänderten Dateien auf.

```
1 git diff
```

 Bash


Zeigt noch nicht indizierte Dateiänderungen an.

```
1 git add [file]
```

 Bash


Indiziert den derzeitigen Stand der Datei für die Versionierung.

```
1 git diff --staged
```

 Bash


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

```
1 git reset [file]
```

 Bash

Nimmt die Datei vom Index, erhält jedoch ihren Inhalt.

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


 Bash

Nimmt alle derzeit indizierten Dateien permanent in die Versionshistorie auf.

## AB Änderungen synchronisieren


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

```
1 git fetch [remote]
```

 Bash


Lädt die gesamte Historie eines externen Repositories herunter.

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

 Bash


Integriert den externen Branch in den aktuell lokal ausgecheckten Branch.

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

 Bash

Pusht alle Commits auf dem lokalen Branch zu GitHub.

```
1 git pull
```

 Bash

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