

# GitLab HowTo

Informatik, Universität Rostock 10.2021

## Inhaltsverzeichnis

<b>1</b>	<b>GIT?</b>	<b>2</b>
1.1	Was was GIT? . . . . .	2
1.2	Was <u>nicht</u> in GIT gespeichert werden sollte . . . . .	2
1.3	Von GIT verwaltete Dateien konfigurieren . . . . .	3
1.3.1	Beispiel: C-Projekt . . . . .	3
1.3.2	Beispiel: TeX-Dokument . . . . .	3
<b>2</b>	<b>Login</b>	<b>4</b>
2.1	Login mit einem Informatikaccount . . . . .	4
2.2	Login mit einem externen Account . . . . .	4
<b>3</b>	<b>SSH-Schlüssel einrichten</b>	<b>5</b>
3.1	Einführung . . . . .	5
3.2	Die Übersichtsseite öffnen . . . . .	5
3.3	Ein SSH-Schlüssel erzeugen . . . . .	6
3.4	Ein SSH-Schlüssel hinzufügen . . . . .	6
3.5	Ein SSH-Schlüssel löschen . . . . .	7
<b>4</b>	<b>Neues Projekt erstellen</b>	<b>7</b>
4.1	Mit der Kommandozeile . . . . .	7
4.2	Mit GITLab-Vorlagen . . . . .	8
<b>5</b>	<b>Repository klonen</b>	<b>10</b>
<b>6</b>	<b>Mit dem Repository arbeiten</b>	<b>11</b>
6.1	Workflow . . . . .	11
6.2	Änderungen untersuchen . . . . .	11
6.3	Arbeitsverzeichnis aktualisieren . . . . .	12
6.4	Arbeitszweig . . . . .	12
6.5	Dinge ändern . . . . .	13
6.6	Änderungen zum Server senden . . . . .	13
6.7	Eine Datei aus der History entfernen . . . . .	13
<b>7</b>	<b>Kontakt</b>	<b>13</b>

# 1 GIT?

## 1.1 Was was GIT?

GIT ist ein dezentrales Versionsverwaltungssystem für die Arbeit von einer oder mehrerer Personen an einem Quelltext. In GIT werden die Veränderungen von Dateien in einer History gespeichert. Korrekt angewendet, können Änderungen an Textdateien zeilenweise nachvollzogen und einem konkreten Nutzer zugeschrieben werden. Falls notwendig, können so ältere Versionen einzelner Dateien wieder hergestellt werden. Da GIT nur die geänderten Zeilen in der History abspeichert, kann es effizient und platzsparend auch eine längere History verwalten. Es lohnt sich also nach jeder noch kleinen Änderung ein Commit zu erstellen, um bei auftretenden Problemen die Änderungen wieder Schrittweise zurück gehen zu können und somit die problematische Stelle zu finden.

In ein GIT Repository gehören:

- Quelltexte
- Makefiles
- Zum Bauen erforderliche Konfigurationen.
- Notwendige Dateien, die nicht generiert oder aus dem Internet / anderen Repositories geladen werden können.
- Dokumentationen mit benötigten Grafiken.
- Protokolle, die ein nicht automatisch reproduzierbaren Prozess dokumentieren.

## 1.2 Was nicht in GIT gespeichert werden sollte

Bei Binärdateien (z.B. Bilder, Videos, ZIP-Dateien, PDF-Dateien, JAR-Pakete) sind die Daten nicht in Textzeilen angeordnet. Deswegen müssen bei Änderungen jeweils die gesamte Datei in der History gespeichert werden. Selbst, wenn sich nur ein Pixel geändert hat! Das führt dazu, dass Binärdateien die Größe eines GIT-Repositories schnell anwachsen lassen können.

Die folgenden Daten sollten deswegen möglichst nicht in GIT gespeichert werden. Insbesondere wenn die Datei > 5 MB ist. (Ausnahme: Die Datei ist für die Erstellung Ihres Projektes erforderlich und lässt sich nicht automatisch generieren.)

- Videos (Z.B: \*.avi ; \*.mp4 ; \*.wmv)
- Bilder (Z.B: \*.jpg ; \*.png ; \*.tiff ; \*.bmp ; \*.xcf ; \*.psd)
- Präsentationen (Z.B: \*.ppt ; \*.pptx ; \*.pdf)
- Ton-Dateien (Z.B: \*.wav ; \*.mp3 ; \*.flac ; \*.ogg)
- Programme, Bibliotheken, Installer (Z.B: \*.exe ; \*.dll ; \*.lib ; \*.msi ; \*.o ; \*.so ; \*.class ; \*.jar)
- Binärdateien (Z.B: \*.bin ; \*.dat ; \*.zip ; \*.gz ; \*.bz)
- Dateien, die sich generieren lassen.
- Dateien, die während des Bauvorganges entstehen. (Z.B: \*.o ; \*.class ; \*.aux ; \*.idx)
- Programm- / Debugausgaben (Z.B: \*.log, \*.dat)
- Ordner anderer Versionsverwaltungssysteme (Z.B: \*.svn)
- Backups, temporäre Dateien, lokale Workspace-Konfigurationen (Z.B: \*.swp ; \*~ ; tmp )

Repositories sollten mit History < 300 MB bleiben. Wenn Sie sich nicht sicher sind, ob Ihre Datei in das GIT-Repository gehört, fragen Sie Ihren Betreuer.

## 1.3 Von GIT verwaltete Dateien konfigurieren

In jedem Ordner des von GIT verwalteten Repositorys kann eine Datei `.gitignore` angelegt werden. Dies ist eine Text-Datei, in der aufgelistet wird, welche Ordner oder Dateien bei einem `git add` ignoriert werden sollen. Dabei kann auch mit Wildcards (\*) gearbeitet werden. „Best practice“ ist, die `.gitignore` Datei nur im Hauptverzeichnis nur im Hauptverzeichnis Ihres Projektes anzulegen.

### 1.3.1 Beispiel: C-Projekt

Das Projekt hat diese Ordner-Struktur:

- Makefile
- `.gitignore`
- `a.out`
- `guessMyNumber.c`
- `guessMyNumber.o`
- `myLibrary.a`
- `myLibrary.c`
- `myLibrary.o`
- `myLibrary.so`

Die `*.c` Dateien sind Quelltexte. Diese Dateien sollten gespeichert werden.

Die `*.o` Dateien sind Compiler-Zwischenergebnisse. Die Datei `a.out` ist das ausführbare Programm nach dem Kompilieren und Linken. Die Dateien `myLibrary.a` und `myLibrary.so` sind aus `myLibrary.c` erstellte Bibliotheken. Diese Dateien sollten nicht in das Repository.

Die `.gitignore` könnte wie folgt aussehen:

```
*.a
*.so
*.o
*.out
```

### 1.3.2 Beispiel: TeX-Dokument

- `.gitignore`
- `myPaper.tex`
- `myPaper.pdf`
- `myPaper.aux`
- `myPaper.idx`
- `myPaper.toc`
- `myPaper.out`
- `myPaper.log`
- `chapter1.tex`
- `chapter1.aux`
- `.chapter1.aux.swp`
- `chapter2.tex`
- `chapter2.tex~`
- `images/`
  - `logo.png`
  - `interestingGraphic.pdf`

Die `*.tex` Dateien sind die Quelltexte zum TeX Dokument. Im Ordner `images` sind für das Dokument wichtige Bilder. Diese Dateien sollten gespeichert werden.

Die Datei `myPaper.pdf` ist das kompilierte Endprodukt. Die `*.log`, `*.out`, `*.aux`, `*.toc`, `*.idx` Dateien sind Nebenprodukte des Bauprozesses. Diese Dateien sollten nicht in das Repository.

Der Editor legt während des Bearbeitens `*.swp` Dateien an (z.B: `chapter1.swp`). Beim Speichern legt der Editor Backups an (z.B: `chapter2.tex~`). Die Backup-Dateien und die `*.swp` Dateien sollen nicht in das Repository.

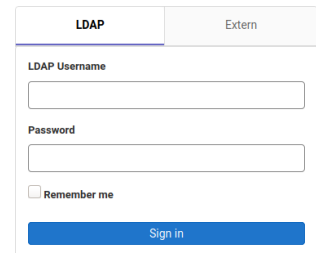
Die `.gitignore` könnte wie folgt aussehen:

```
*.swp
*~
*.aux
*.idx
*.toc
*.out
*.log
```

## 2 Login

### 2.1 Login mit einem Informatikaccount

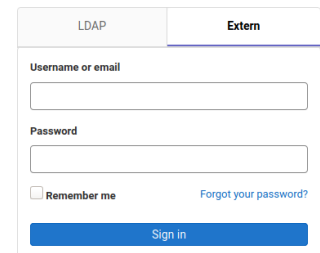
1. Navigiere im Browser zu:  
[https://git.informatik.uni-rostock.de/users/sign\\_in](https://git.informatik.uni-rostock.de/users/sign_in)
2. Wählen sie: LDAP
3. Geben Sie das Kürzel Ihres Informatik-Accounts ein.
4. Geben Sie das Passwort Ihres Informatik-Accounts ein.
5. Klicken Sie auf Sign In.



The screenshot shows a login form with two tabs: 'LDAP' (selected) and 'Extern'. The 'LDAP' tab contains the following elements: a text input field labeled 'LDAP Username', a text input field labeled 'Password', a checkbox labeled 'Remember me', and a blue button labeled 'Sign in'.

### 2.2 Login mit einem externen Account

1. Navigiere im Browser zu:  
[https://git.informatik.uni-rostock.de/users/sign\\_in](https://git.informatik.uni-rostock.de/users/sign_in)
2. Wählen sie: Extern
3. Geben Sie Ihre E-Mail-Adresse ein.
4. Geben Sie Ihr Passwort ein.
5. Klicken Sie auf Sign In.



The screenshot shows a login form with two tabs: 'LDAP' and 'Extern' (selected). The 'Extern' tab contains the following elements: a text input field labeled 'Username or email', a text input field labeled 'Password', a checkbox labeled 'Remember me', a link labeled 'Forgot your password?', and a blue button labeled 'Sign in'.

## 3 SSH-Schlüssel einrichten

### 3.1 Einführung

SSH-Schlüssel werden zur Authentifizierung gegenüber dem GIT Server verwendet. Diese bestehen aus einem Paar an öffentlichem und privatem Schlüssel. Der öffentliche Schlüssel wird dem GIT-Server mitgeteilt. Der private Schlüssel wird geheim gehalten. Um im Schadensfall (z.B. der Zugang auf einem Rechner wurde gehackt) gezielt einzelne Rechner oder Nutzer sperren zu können, sollte dieses Schlüsselpaar für jeden Nutzer und Rechner einzigartig sein.

### 3.2 Die Übersichtsseite öffnen

- Klicken Sie oben rechts auf Ihr Symbol.
- Klicken Sie auf `Edit profile` in dem erscheinenden Menü.
- Klicken Sie auf `SSH Keys` im Menü auf der linken Seite.

The screenshot displays the GitLab user interface for managing SSH keys. On the left, the 'User Settings' sidebar is visible, with 'SSH Keys' highlighted. The main content area is titled 'SSH Keys' and contains the following elements:

- Search settings:** A search bar for finding specific settings.
- SSH Keys:** A section explaining that SSH keys allow for a secure connection between the user's computer and GitLab.
- Add an SSH key:** A section with instructions on how to add a new key, including a text input field for the public key (with a hint: 'Typically starts with "ssh-ed25519 ..." or "ssh-rsa ..."').
- Title and Expires at:** Two input fields for configuring the key's title and expiration date.
- Your SSH keys (2):** A list of existing keys, each with a title, creation date, last used date, and expiration status.

### 3.3 Ein SSH-Schlüssel erzeugen

Dieser Schritt ist erforderlich, wenn sie noch keinen SSH-Schlüssel haben oder ein neuen Schlüssel für GIT einrichten wollen. Andernfalls können Sie diesen Schritt überspringen.

- Starten Sie eine Kommandozeile auf ihrem Computer.
- Führen Sie das folgende Kommando aus:

```
ssh-keygen -t ecdsa -b 521
```

Die Ausgabe sieht so ähnlich aus wie:

```
[bob@earth ~]$ ssh-keygen -t ecdsa -b 521
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_ecdsa
Your public key has been saved in /home/bob/.ssh/id_ecdsa.pub
The key fingerprint is:
SHA256:5vKu...9Jk0 bob@earth
The key's randomart image is:
+---[ECDSA 521]---+
|                 .. |
|                 .. |
|                o  . |
|               o . . +..o |
|            . = . S +o.+ . |
|         ....+ +   .E o |
|      +.*..+ o  o + . |
|   . Booo.+ o. o + |
| o..oooo+ o=++  |
+-----[SHA256]-----+
```

- Auf die Frage „Enter file in which to save the key“ können Sie einfach **[Enter]** drücken und die Vorgabe übernehmen oder einen eigenen Dateinamen angeben.
- Auf die Aufforderung „Enter passphrase“ können Sie zur zusätzlichen Sicherheit ein Passwort für ihr SSH-Schlüssel festlegen. Ihre Eingabe müssen Sie in der Zeile „Enter same passphrase again“ wiederholen. Das Passwort wird während der Eingabe nicht angezeigt. Dieses Passwort ist bei jedem Senden oder Empfangen von Daten zum oder vom GIT-Server einzugeben.


### 3.4 Ein SSH-Schlüssel hinzufügen

- Sie finden Ihre SSH-Schlüssel in Ihrem Home-Verzeichnis, im Ordner `.ssh`. Die für diesen Schritt interessanten Dateien haben die Endung `.pub`. Z.B: `id_ecdsa.pub` oder `id_rsa.pub` Falls Sie im vorherigen Schritt ein SSH-Schlüssel angelegt haben wurde Ihnen in der Zeile „Your public key has been saved in ...“ der Dateiname der gesuchten Datei angegeben.
- Öffnen sie die Datei mit Ihrem public key in einem Text-Editor. Der Inhalt ist eine einzige Zeile und sollte etwa wie folgt aussehen:

```
ecdsa-sha2-nistp521 AAAA...FHw== user@computer
```

- Kopieren Sie den Schlüssel in das Feld `key`.
- Geben Sie dem Schlüssel einen Titel (Feld: `Titel`), mit dem Sie wiedererkennen können, zu welchem Account/Computer dieser gehört.
- Klicken Sie auf `Add key`.

### 3.5 Ein SSH-Schlüssel löschen

- Finden Sie in der Liste `Your SSH keys` den Schlüssel, den Sie löschen wollen.
- Klicken Sie auf das Mülltonnensymbol , rechts neben den Schlüssel.
- Bestätigen Sie die Frage `Are you sure you want to delete this SSH key?` mit Klick auf `Delete`.

## 4 Neues Projekt erstellen

### 4.1 Mit der Kommandozeile

- **Sie haben noch kein lokales Projekt?**  
Verwenden Sie alle Schritte (1) bis (6)
- **Sie haben ein lokales Projekt, aber es ist nicht mit GIT verwaltet?**  
Verwenden Sie Schritte (2), (4), (5) und (6)
- **Sie haben ein lokales GIT-Projekt?**  
Verwenden Sie Schritte (2) und (6)

Für das Beispiel wird der Account **myAccount** und der Projekt-Name **myProject** angenommen. Die **grünen** Teile der Beispiele müssen durch die zu Ihnen passenden Werte ersetzt werden.

1. Ordner für das Projekt erstellen:

```
mkdir myProject
```

2. Ordner für das Projekt betreten:

```
cd myProject
```

3. Einige Dateien anlegen:

```
echo 'My Project' > README.md
```

4. GIT initialisieren.

```
git init
```

5. Die Dateien dem neuen GIT-Repository hinzufügen:

```
git add .
```

```
git commit -m "Initial commit."
```

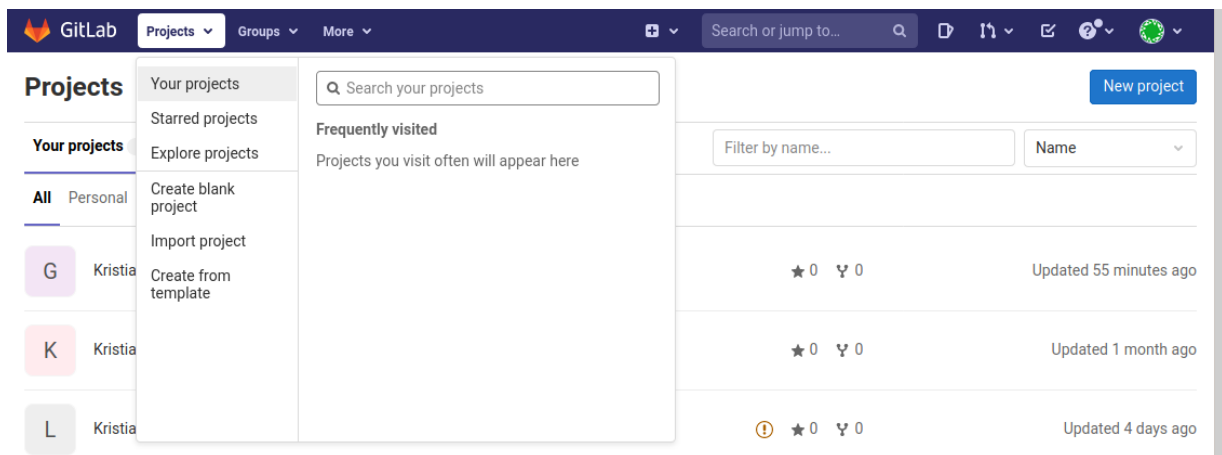
6. Projekt zum Server senden.

```
git push -set-upstream git@git.informatik.uni-rostock.de:myAccount/myProject master
```

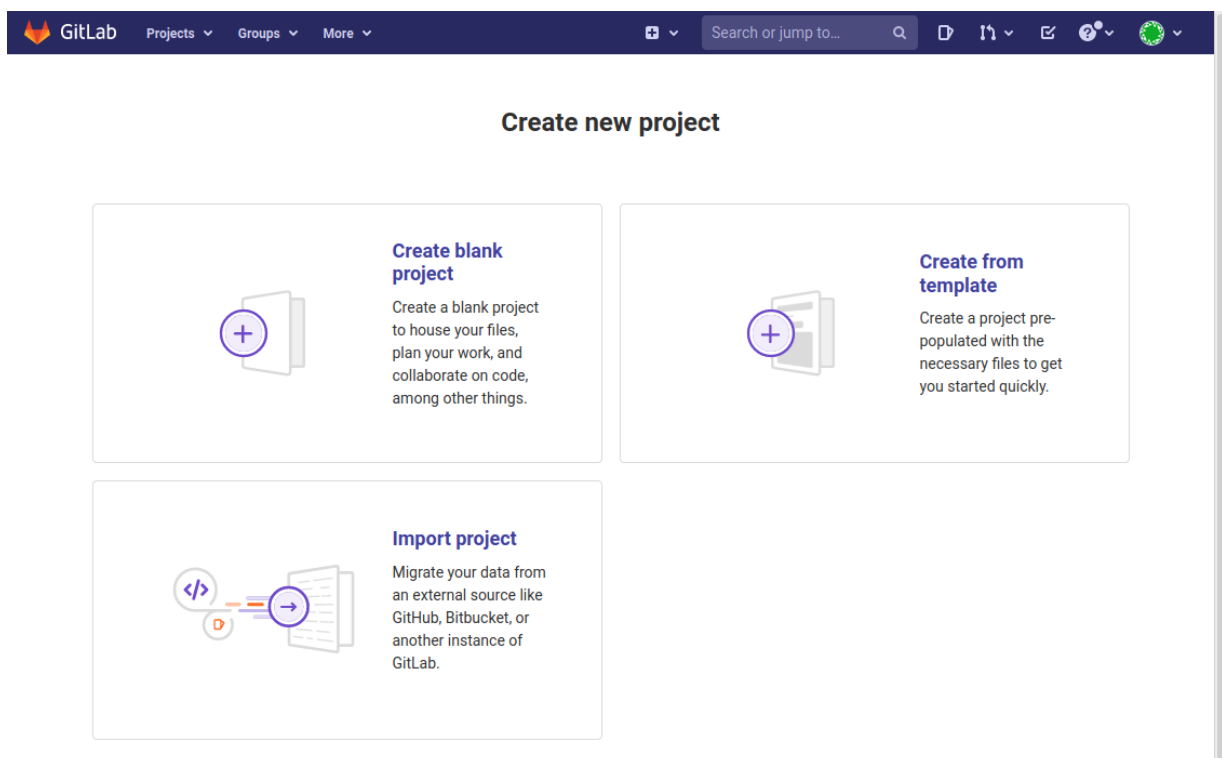


## 4.2 Mit GITLab-Vorlagen

- Klicken Sie oben links auf `Projects`.  
Es erscheint das Projektauswahlmenü.



- Im darauf erscheinenden Menü klicken Sie auf `your projects`  
Es erscheint die Liste Ihrer Projekte.
- Klicken Sie auf `new project` am rechten Rand.  
Es erscheint die Projekt-Erstellen-Auswahl.



- Klicken Sie auf `Create blank project`.

The screenshot shows the GitLab interface for creating a new project. The header includes the GitLab logo and navigation menus. The main content area is titled 'New project - Create blank project'. On the left, there is a 'Create blank project' section with a plus icon and a brief description. The main form contains the following fields and options:

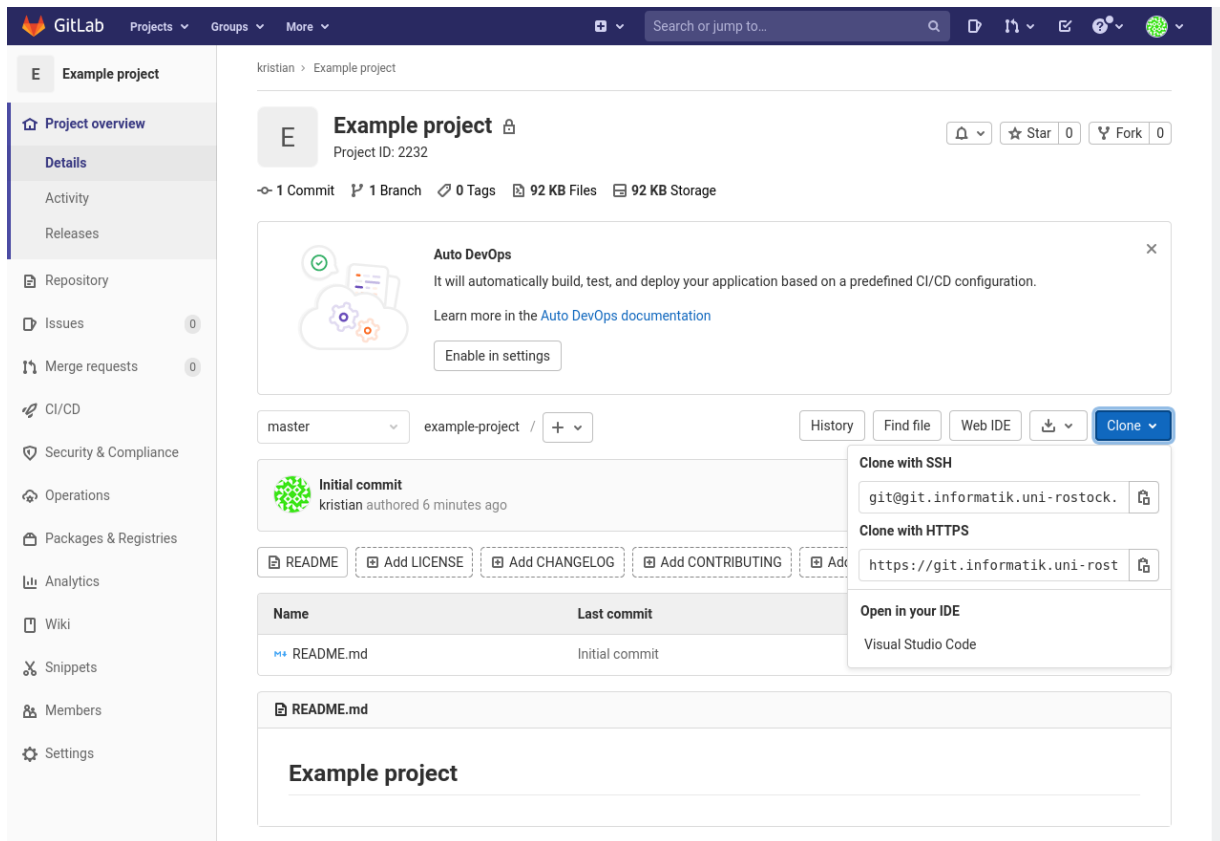
- Project name:** A text input field with the placeholder 'Example project'.
- Project URL:** A text input field with the placeholder 'https://git.informatik.uni-rostock.de/my-account-name'.
- Project slug:** A text input field with the placeholder 'example-project'.
- Project description (optional):** A large text area with the placeholder 'Description format'.
- Visibility Level:** Three radio button options: 'Private' (selected), 'Internal', and 'Public'.
- Initialize repository with a README:** A checked checkbox.

At the bottom of the form, there is a blue 'Create project' button and a grey 'Cancel' button.

- Geben Sie einen Namen für Ihr Projekt im Feld `Project name` ein.
- Sie können eine kurze Beschreibung Ihres Projektes im Feld `Project description` angeben.
- Aktivieren Sie das Häkchen `Initialize repository with a README`, damit Sie ein nicht leeres Repository zum Auschecken haben.
- Klicken Sie auf `Create project`.  
Es erscheint die Übersichtsseite des neuen Projektes.  
Nächster Schritt ist, Ihr neues Repository auf Ihren Computer zu klonen. (Kapitel 5)

## 5 Repository klonen

- Öffnen Sie Ihr Projekt auf <https://git.informatik.uni-rostock.de>.
- Klicken Sie auf **Clone** und kopieren Sie den Link unter **Clone with SSH**.



- Öffnen Sie ein Terminal.
- Navigieren Sie zum Ordner, in dem Sie das Repository speichern wollen.

```
cd my/favorite/repo/storage/directory
```

- Ersetzen Sie im unten stehenden Befehl `git@git.informatik.uni-rostock.de:example-user/example-project.git` durch den eben kopierten Link und führen Sie den Befehl aus.

```
git clone git@git.informatik.uni-rostock.de:example-user/example-project.git
```

Falls GIT in diesem Schritt nach einem Login fragt, fehlt die SSH Konfiguration für diesen Rechner oder Account in GITLab. Siehe Kapitel 3: „SSH-Schlüssel einrichten“.

## 6 Mit dem Repository arbeiten

### 6.1 Workflow

Beim Bearbeiten von Daten in einem GIT-Repository wird in den meisten Fällen ein Workflow wie der hier vorgestellte Workflow verwendet.

1. Die aktuellsten Änderungen vom Server holen. fetch / pull
2. Falls notwendig: Den aktuellen Arbeitszweig wechseln oder einen neuen anlegen. checkout / branch
3. Dateien hinzufügen, ändern oder löschen. add / remove
4. Änderungen mit einem sinnvollen Kommentar committen. commit
5. Neueste Commits zum Server senden. push

### 6.2 Änderungen untersuchen

- `git status`

Zeigt die lokalen Änderungen zum letzten committeten Stand im aktuellen Arbeitszweig.

Beispiel:

```
[bob@earth ~/myProject] (master)$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   main.c

no changes added to commit (use "git add" and/or "git commit -a")
```

- `git log`

Zeigt eine Liste mit den Commits.

Beispiel:

```
[bob@earth ~/myProject] (master)$ git log
commit 1c5e59568e86377b620614f9e48250b4181fe9ab
Author: MySelf <my-self@my-server.de>
Date: Tue May 4 17:10:13 2021 +0200

    Added Ackermann function.

commit 899efba3143f3c24fa9168c803f2caf5d4fe3974
Author: MySelf <my-self@my-server.de>
Date: Tue May 4 13:13:46 2021 +0200

    First commit.
```

## 6.3 Arbeitsverzeichnis aktualisieren

- `git fetch`  
Lädt den aktuellsten Stand des Repositorys vom Server lässt das Arbeitsverzeichnis aber unverändert.
- `git pull`  
Lädt den aktuellsten Stand des Repositorys vom Server und aktualisiert das Arbeitsverzeichnis.
- `git checkout NAME`  
Aktualisiert die lokale Arbeitskopie des Repository entsprechend des Wertes von `NAME`.
  - Falls `NAME` eine der Dateien im Repository ist, dann wird diese Datei auf den Stand des letzten Commits zurück gesetzt.
  - Falls `NAME` die ID eines Commits ist, dann werden alle Dateien durch ihre Versionen in diesem Commit ersetzt. Im Beispiel von `git log` von oben:

```
git checkout 899efba3143f3c24fa9168c803f2caf5d4fe3974
```

Kehrt zum Stand des Commits mit dem Titel `First commit.` zurück.

- Falls `NAME` der Name eines Arbeitszweiges ist, dann werden alle Dateien durch ihre Version aus diesem Arbeitszweig ersetzt. Alle weiteren Commits werden zu diesem Arbeitszweig hinzugefügt.

## 6.4 Arbeitszweig

Arbeitszweige sollten immer dann angelegt werden, wenn Sie vorhaben neue Dinge Ihrem Projekt hinzuzufügen oder größere Änderungen vorzunehmen. Wenn Sie Ihre Änderungen durchgeführt und erfolgreich getestet haben, können Sie den Arbeitszweig wieder mit dem Hauptzweig vereinigen.

- `git branch`  
Zeigt den aktuellen und die auf diesem Computer verfügbaren Arbeitszweige an.
- `git branch myNewFeature`  
Legt einen neuen Arbeitszweig mit dem Namen `myNewFeature` an. Es werden die Daten aus dem aktuellen Arbeitszweig kopiert.
- `git checkout myOtherNewFeature`  
Wechselt zum Arbeitszweig mit dem Namen `myOtherNewFeature`.
- `git merge myNewFeature`  
Vereinigt den Arbeitszweig `myNewFeature` in den aktuellen Arbeitszweig.

Beispiel: Der aktuelle Arbeitszweig ist `master`. Es soll die Datei `example.c` hinzugefügt und die Datei `main.c` bearbeitet werden.

```
[bob@earth ~/myProject] (master)$ git pull
[bob@earth ~/myProject] (master)$ git branch example
[bob@earth ~/myProject] (example)$ vim example.c
[bob@earth ~/myProject] (example)$ git add example.c
[bob@earth ~/myProject] (example)$ vim main.c
[bob@earth ~/myProject] (example)$ git add main.c
[bob@earth ~/myProject] (example)$ git commit -m "Added new example function."
[bob@earth ~/myProject] (example)$ git push

[bob@earth ~/myProject] (example)$ git checkout master
[bob@earth ~/myProject] (master)$ git pull
[bob@earth ~/myProject] (master)$ git merge example
[bob@earth ~/myProject] (master)$ git push
```

## 6.5 Dinge ändern

- `git add FILE_NAME`  
Fügt die neue oder geänderte Datei `FILE_NAME` dem nächsten Commit hinzu.
- `git add .`  
Fügt alle neuen oder geänderten Dateien im aktuellen Ornder und allen unterordnern dem nächsten Commit hinzu.
- `git rm FILE_NAME`  
Entfernt die Datei `FILE_NAME` aus der Arbeitskopie und markiert sie als gelöscht im nächsten Commit.
- `git commit`  
Speichert die vorgemerkten Änderungen in einem Commit. Es wird ein Texteditor angezeigt, in dem ein Kommentar zu diesem Commit angegeben werden muss.

## 6.6 Änderungen zum Server senden

Wenn Sie Ihre Änderungen committet haben, dann sollten sie diese anschließend zum Server senden:

```
git push
```

## 6.7 Eine Datei aus der History entfernen

- Es dürfen keine offenen Merge requests sein.
- Dieses Kommando entfernt die Datei aus Ihrer lokalen History:

```
git filter-branch -force \  
  -index-filter "git rm -cached -ignore-unmatch FILE_NAME" \  
  -prune-empty -tag-name-filter cat - -all
```

- Fügen Sie die Datei zur `.gitignore` Datei hinzu.

```
echo "FILE_NAME" >> .gitignore
```

```
git commit -m "Add FILE_NAME to .gitignore"
```

- Dieser befehl sendet die geänderte History zum Server.

```
git push origin -force -all
```

- Sagen Sie Ihren Mitarbeitern, dass sie alle Branches, die sie aus Ihrem alten Repository-Verlauf erstellt haben, neu aufbauen (rebase) und nicht zusammenführen (merge) sollen. Ein Merge-Commit könnte die alte Historie oder Teile davon wieder einfügen.

Details zum Thema „rebase“: <https://git-scm.com/book/en/Git-Branching-Rebasing>

- Weitere Informationen zum Thema „Daten aus der History Löschen“ finden Sie unter:  
<https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository>

## 7 Kontakt

Bei Anregungen und Fragen senden Sie bitte eine E-Mail an: [stg-cs@uni-rostock.de](mailto:stg-cs@uni-rostock.de)