

Umgang mit Git am lokalen PC

Inhalt

1. Klienten	2
1.1. GitButler	3
1.1.1. Repository eröffnen/ klonen	3
1.1.2. Hauptansicht	4
1.1.3. Mergekonflikte	6
1.2. Ungit	7
1.2.1. Installation	7
1.2.2. Repository eröffnen	7
1.2.3. Commit	7
1.2.4. Push & Pull (Arbeiten mit Remote-Verzeichnissen)	8
1.2.5. Checkout / Historie	9
1.2.6. Merge	9
1.3. SmartGit	10
1.3.1. Installation	10
1.3.2. Repository eröffnen	12
1.3.3. Repository klonen	14
1.3.4. Übersichten	15
1.3.5. Mergen und Mergekonflikte	15
2. Git-Konsole	19
2.1. Anlegen eines Repositories	19
2.1.1. Konsole starten	19
2.1.2. Navigation	19
2.1.3. Initialisierung	19
2.1.4. Verzeichnis prüfen	19
2.2. Clonen eines Repositories	20
2.2.1. Konsole starten	20
2.2.2. Navigation	20
2.2.3. Klonvorgang anstoßen	20
2.2.4. Erfolg prüfen	20
2.3. Stage / Commit	20
2.3.1. Überprüfen der Änderungen	20
2.3.2. Stagen der Änderungen	20
2.3.3. Überprüfen der gestagten Änderungen	21
2.3.4. Committen der Änderungen	21

2.3.5. Überprüfen des Commit-Erfolgs	21
2.4. Check out	21
2.4.1. Überprüfen des Repository-Status	21
2.4.2. Aus-checken des Gewünschten Branchs oder Commits	22
2.4.3. Überprüfen des Wechsel-Erfolgs	22
2.5. Push	22
2.5.1. Überprüfen des Repository-Status	22
2.5.2. Pushen der Änderungen	22
2.5.3. Authentifizierung (falls erforderlich)	23
2.5.4. Überprüfen des Push-Erfolgs	23
2.6. Pull	23
2.6.1. Pullen der neuesten Änderungen	23
2.6.2. Authentifizierung (falls erforderlich)	23
2.6.3. Überprüfen des Pull-Erfolgs	23
2.7. Merge / Rebase	23
2.7.1. Navigieren zum Ziel-Branch	24
2.7.2. Mergen des Quell-Branchs	24
2.7.3. Bearbeiten von Merge-Konflikten (falls erforderlich)	24
2.8. Branches	24
2.8.1. anlegen	24
Überprüfen des Repository-Status	24
Anlegen des Neuen Branchs	25
Wechseln zum Neuen Branch	25
Überprüfen des Branch-Erfolgs	25
2.8.2. mergen	25
Überprüfen des Repository-Status	25
Wechseln zum Ziel-Branch	25
Mergen des Quell-Branchs	26
Bearbeiten von Merge-Konflikten (falls erforderlich)	26
Überprüfen des Merge-Erfolgs	26

1. Klienten

Zum Zeitpunkt des Erstellens dieser Materialien bieten sich verschiedene Lösungen für Desktopanwendungen an. Beispiele wären hier:

- GitButler, welches unter <https://gitbutler.com/> erreichbar ist und open source eingesehen werden kann.
- Ungit ist unter <https://github.com/FredrikNoren/ungit> bzw. <https://sourceforge.net/projects/ungit.mirror/> verfügbar und ebenfalls open source. Es erfordert die Installation von [Node.js](#).

- SmartGit ist ein kommerzielles Produkt aus dem Hause Syntevo. Die Projektseite findet sich unter <https://www.syntevo.com/smartgit/>

1.1. GitButler

Nach der Installation findet man sich im Startfenster von GitButler.



Abb. 1. Dialogfeld Neuanlegung

1.1.1. Repository eröffnen/ klonen

GitButler kann zum Zeitpunkt der Erstellung dieses Dokuments Repositories nur verwalten, nicht frisch anlegen. Daher wäre der Ablauf der, dass man zunächst ein Repository in der Webmaske erstellt und dieses dann via "Clone repository" auf den lokalen Rechner klonet. GitButler tut sich erfahrungsgemäß schwer damit leere Repositories zu klonen. Um dies zu umgehen kann man eine leere Textdatei in der Online-Ansicht des Repositories anlegen.

Clone a repository

Clone URL

`https://fortbildner.gitcamp-bw.de/Patrick.Gerth/Testrepo1.git`

Where to clone

`C:\Users\Patri\Desktop\Testordner`

Choose..

Cancel

Clone >

Abb. 2. Dialogfeld Klonen

Im Anschluss öffnet sich die Hauptansicht.

1.1.2. Hauptansicht

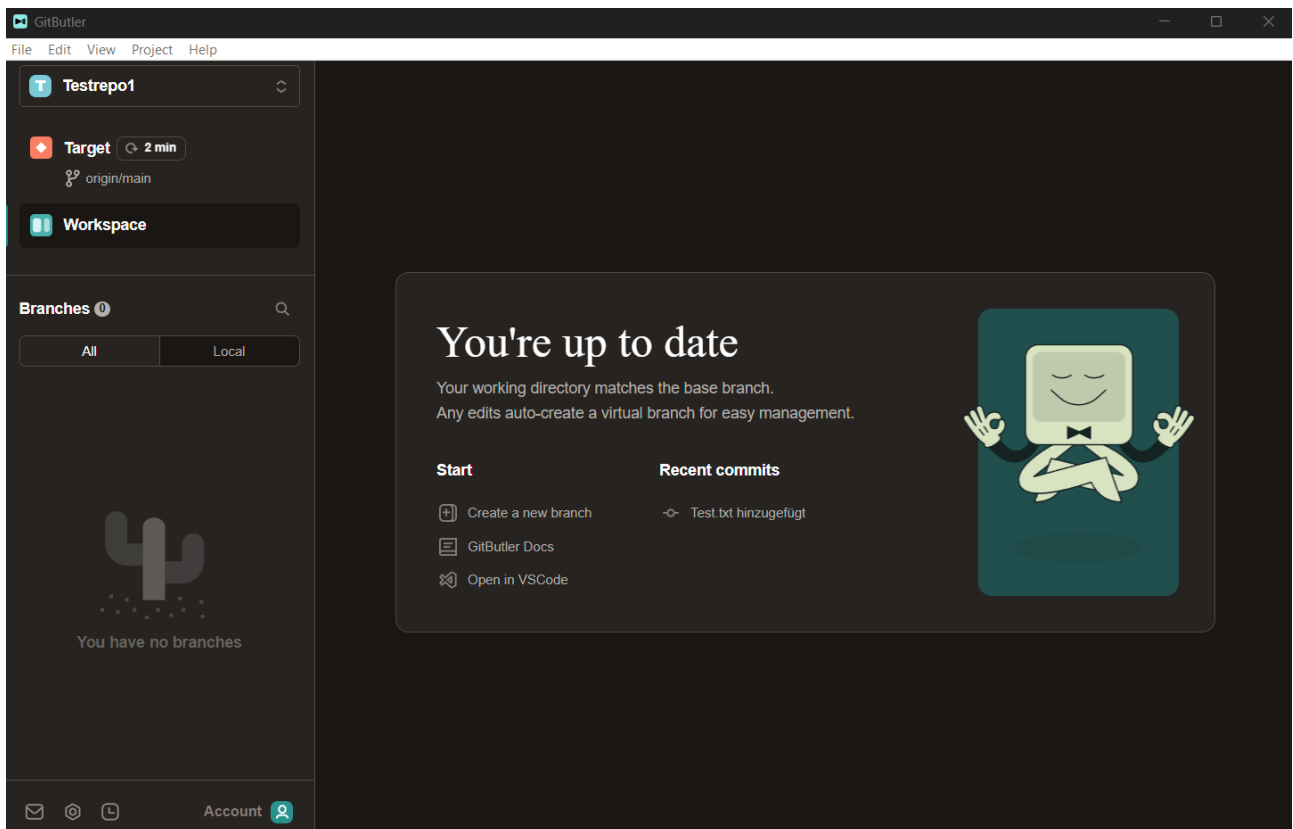


Abb. 3. Hauptansicht

Hier lassen sich verschiedene Branches anlegen, in welche die entsprechenden Dateien per drag-and-drop hineingelegt werden können:

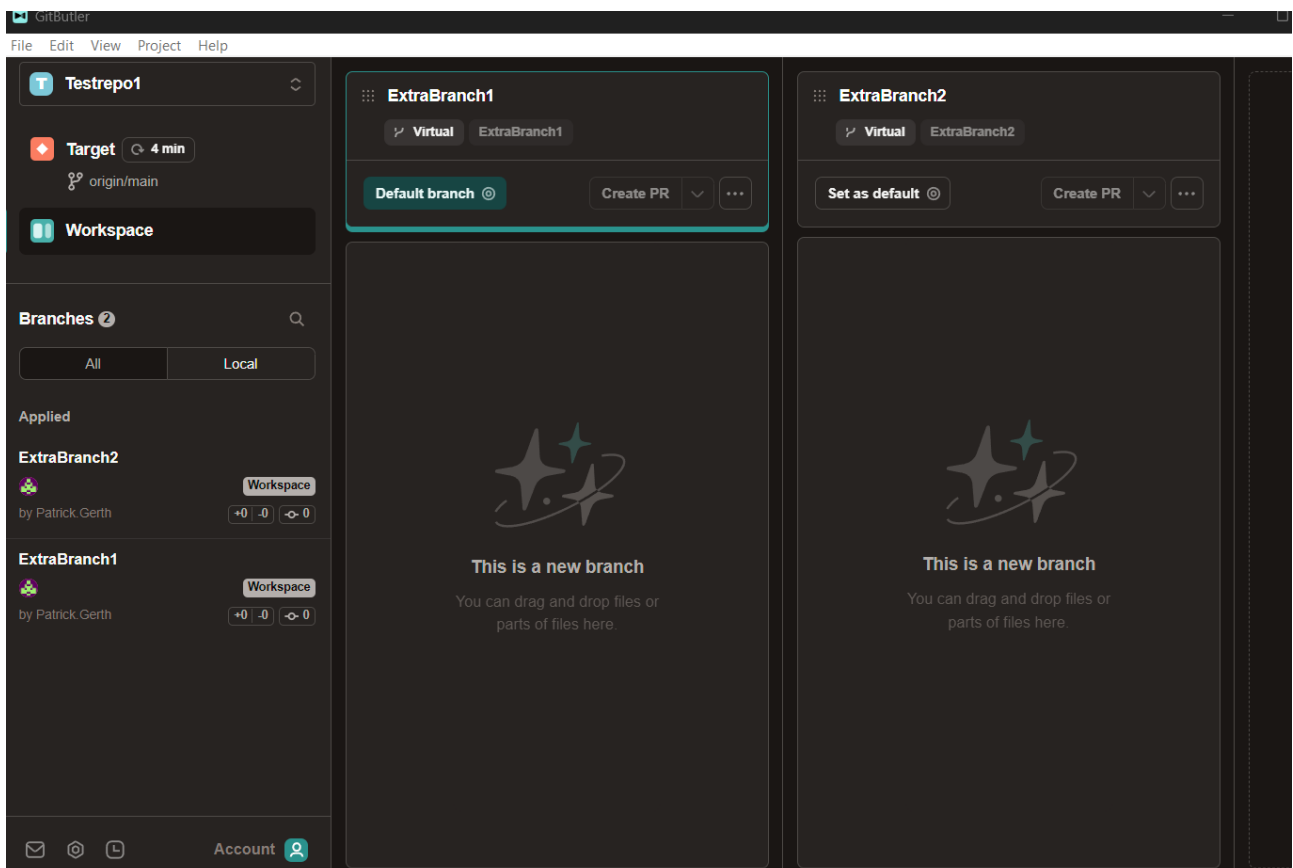


Abb. 4. Branch-Ansicht

Für eine detaillierte Führung durch die Benutzung empfiehlt sich ein Blick in die Dokumentation unter <https://docs.gitbutler.com/>

1.1.3. Mergekonflikte

In seiner aktuellen Fassung (Stand 10.09.2024) verfügt Gitbutler nur über sehr eingeschränkte Möglichkeiten Mergekonflikte zu lösen. Statt dessen wird jedes mal lokal ein neuer Entwicklungsbranch (Virtualbranch) angelegt, welcher dann per Konsole oder Webanwendung wieder gemerged werden muss.

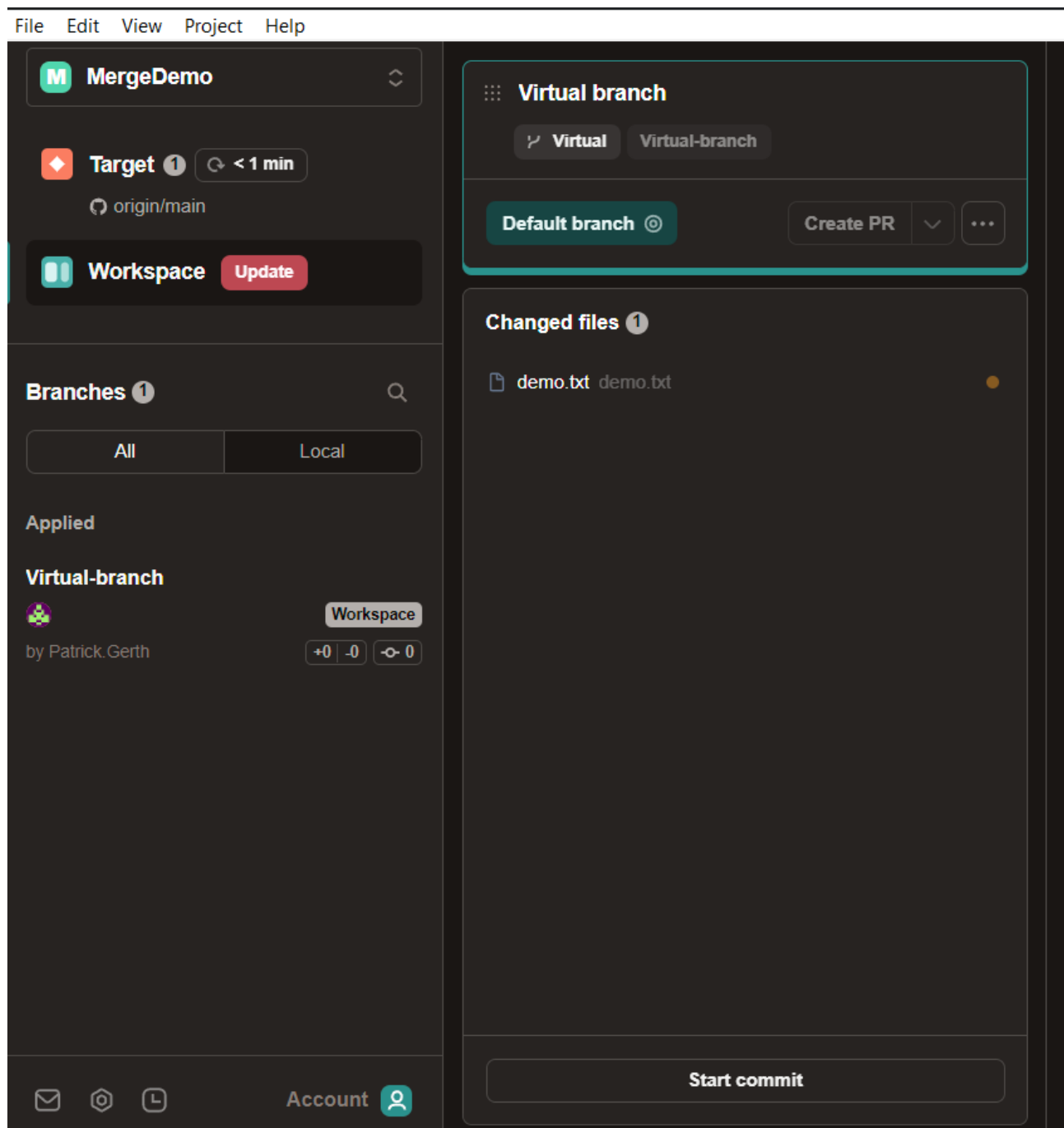


Abb. 5. Branch-Ansicht mit Virtual branch

1.2. Ungit

Ungit kommt mit einer schlanken, übersichtlichen Benutzeroberfläche, die sich an Anfänger in der Versionsverwaltung richtet. Etwas ungewöhnlich, aber intuitiv ist, dass man viele Aktionen durch Drag & Drop von Branches ausführt.

1.2.1. Installation

Für die Installation von ungit werden [Node.js](#) sowie git ($\geq 2.34.x$) benötigt. Um ungit zu installieren, geben Sie nach der Installation von Node.js den folgenden Befehl ein:

```
npm install -g ungit
```

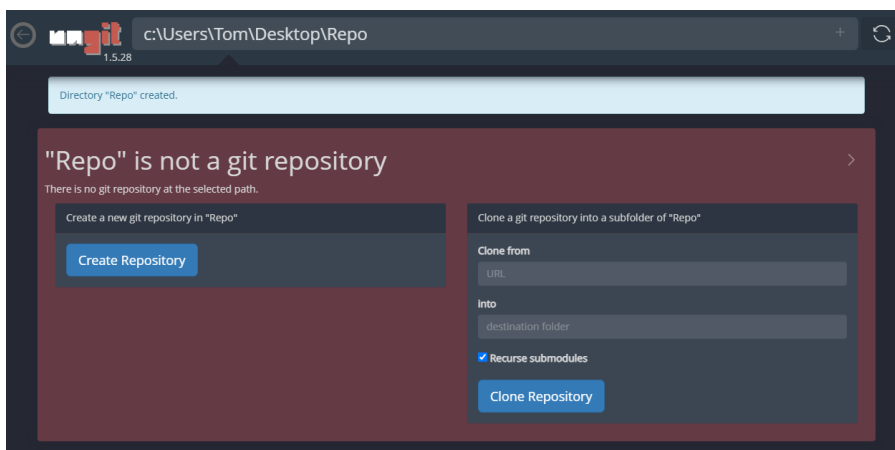
Falls Ihr System Root-Rechte für die Installation globaler npm-Pakete erfordert, sollten Sie den Schalter **-H** verwenden:

```
sudo -H npm install -g ungit
```

Alternativ stehen vorkompilierte Pakete zur [Verfügung](#); git wird jedoch auch in diesem Fall weiterhin benötigt.

1.2.2. Repository eröffnen

Man startet immer mit der Auswahl eines Verzeichnisses in der Kopfzeile. Gegebenfalls wird das Verzeichnis neu angelegt. Handelt es sich noch nicht um ein Git-Repository, wird ein neues angelegt, indem es initialisiert wird oder ein bestehendes online-Repository geklont wird.

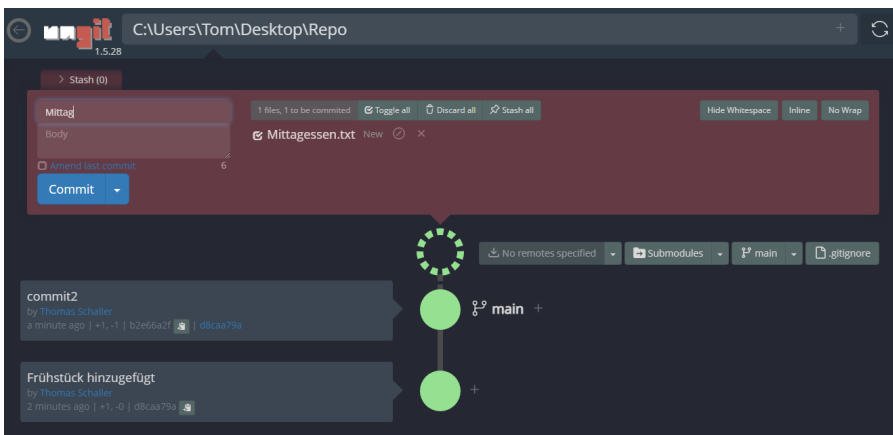


Bemerkung: Beim Clonen kann man leider nicht die Bezeichnung des Remotes wählen.

1.2.3. Commit

Bei jeder Änderung des Repos wird automatisch ein Fenster für den Commit aller geänderten Dateien angeboten. Das Stagen der Dateien entfällt. Dies ist für Anfänger gut geeignet, da sie

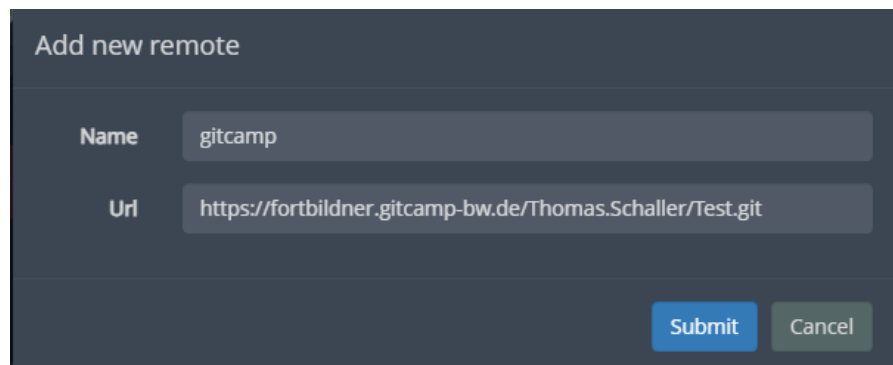
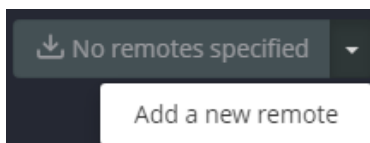
ohnehin keine Auswahl der zu versionierenden Dateien treffen. Alternativ kann man Dateien aber durch einen einzigen Klick zu gitignore hinzufügen.



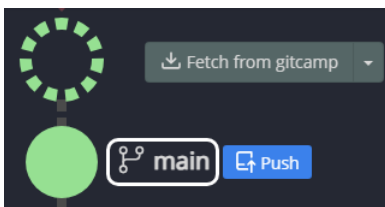
Die Darstellung der Commit-History ist sehr anschaulich.

1.2.4. Push & Pull (Arbeiten mit Remote-Verzeichnissen)

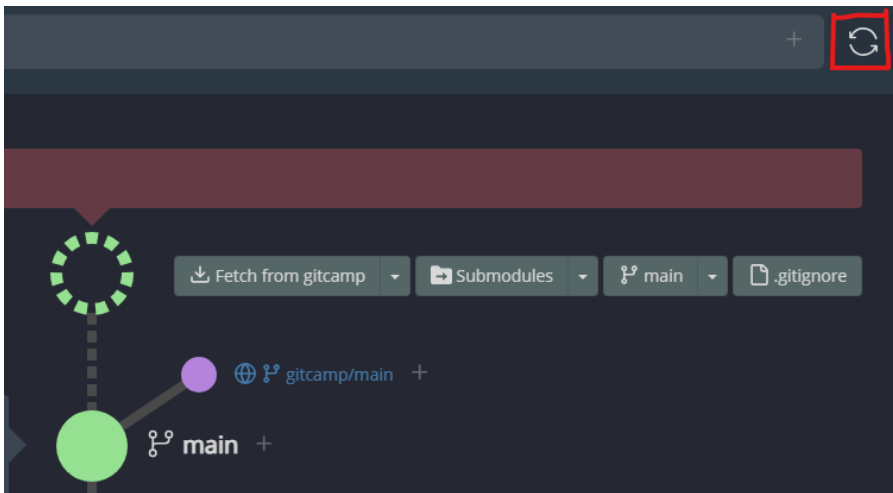
Existiert noch keine Anbindung an ein Gitcamp-Repository muss die Verbindung zunächst eingerichtet werden. Klont man ein Repo passiert das automatisch.



Anschließend kann man durch Anklicken des aktuellen Branches ("Fetch from Gitcamp" → Gitcamp ist ausgewählt) die Dateien auf das gewählte Remote pushen.



Gibt es Änderungen im Remote-Repo dann macht man diese durch "Refresh Changes" sichtbar.



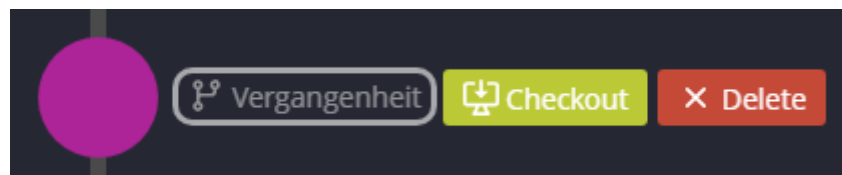
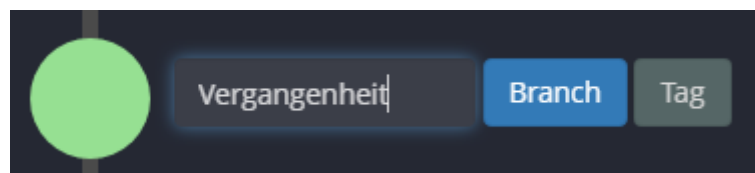
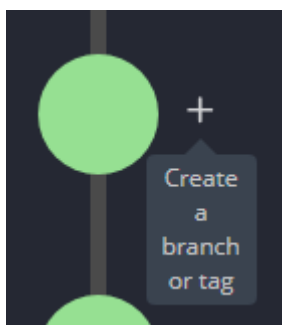
Einen Pull-Button gibt es leider nicht, aber man kann den Branch durch Drag & Drop auf den remote-Branch verschieben. Die Dateien werden dann automatisch per git pull aktualisiert.



Umgekehrt kann man eine push-Operation auch durch das Verschieben des Remote-Repo-Zeigers erreichen.

1.2.5. Checkout / Historie

Möchte man zu älteren Versionen wechseln, ist es in ungit nicht möglich nur den HEAD zu verschieben. Es wird immer ein Branch benötigt. Das vermeidet aber den problematisch detached head-State.

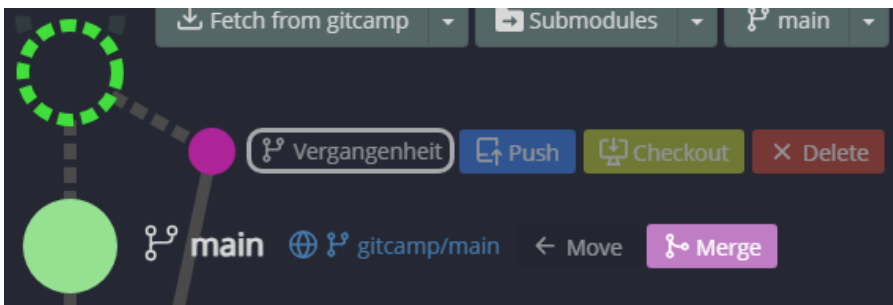


Den Zeiger auf den "Vergangenheit"-Branch kann man nun aber beliebig in der Commit-Historie per Drag & Drop verschieben.

1.2.6. Merge

Einen anderen Branch merged man zum aktuellen Branch hinzu, indem man den anderen

Branch-Zeiger anklickt und dann den "Merge"-Button drückt. Schön ist, dass der Effekt der Operation visuell dargestellt wird. Gleiches gilt für die Rebase-Operation.



Bei Merge-Konflikten wird leider kein automatischer Aufruf eines Merge-Tools vorgenommen. Das muss in der Konsole durch den Befehl `git mergetool` erfolgen.



Um **meld** ([Download](#)) dauerhaft als Mergetool zu konfigurieren, wird folgende Git-Konfiguration gesetzt:

```
git config --global merge.tool meld
git config --global mergetool.keepBackup false
```

Damit wird **meld** als Standard-Mergetool verwendet und es werden keine Sicherungskopien der Dateien angelegt.

1.3. SmartGit

1.3.1. Installation

Smartgit gibt es in unterschiedlichen Varianten (<https://www.syntevo.com/register-non-commercial/#academic>): eine kostenpflichtige Version, eine kostenlose Education-Version und eine kostenlose Privater (Hobby-Edition). Für einfache Anwendungen (ohne kollaboratives Arbeiten) reicht die Hobby-Edition. Diese können Schülerinnen und Schüler leicht zu Hause installieren. In der Schule empfiehlt sich die Education-Version. Dafür muss die Software mit einer E-Mail-Adresse der Schule registriert werden. Die Lizenzdatei (nach Absprache mit der Firma) darf dann von allen Lehrern und Schülern der Schule benutzt werden.

Installation unter Windows: Die Software kann als portable Version auf dem Server der Schule abgelegt werden, wenn einige zusätzliche Einstellungen vorgenommen werden:

1. Die Datei `smartgit\bin\smartgit.voptions` muss so angepasst werden, dass die Settings-Einstellungen im Homeverzeichnis der Schüler gespeichert werden:

```
-Dsmartboot.sourceDirectory=H:\smartgit-settings\updates
-Dsmartgit.settings=H:\smartgit-settings
-XX:ErrorFile=%EXE4J_EXEDIR%..\settings\hs_err_pid%p.log
```

2. Dieses Verzeichnis muss beim ersten Starten von Smartgit mit den Standardwerten gefüllt

werden (insbesondere der Lizenzdatei und den Proxy-Einstellungen). Daher führt man Smartgit als Admin einmalig aus und konfiguriert die Lizenzdatei und die Starteinstellungen. Die kompletten Einstellungen werden in dem Verzeichnis `.settings` bzw. `H:\smartgit-settings` (wenn man Punkt 1 schon durchgeführt hat) gespeichert.

Dieses Verzeichnis kopiert man auf dem Server in ein eigenes Verzeichnis z. B. `smartgit-settings-schulexy`.

3. Proxy-Einstellungen: Damit der Proxy der Schule benutzt wird, muss man in der Datei `preferences.yml` folgende Zeilen anpassen:

```
proxy: {user: %user%, authenticate: true, host: 10.10.0.1, enabled: true, port:
8080,
  autoDetect: false}
```

Die Variable `%user%` wird später durch ein Skript durch den Usernamen ersetzt.

4. Statt direkt `smartgit.exe` zu starten, muss die Verknüpfung so angelegt werden, dass eine Batch-Datei mit folgendem Inhalt gestartet wird:

```
if exist H:\smartgit-settings\ (
  echo Settingsverzeichnis schon vorhanden
) else (
  mkdir H:\smartgit-settings
  xcopy P:\informatik\smartgit\smartgit-settings-schulexy H:\smartgit-settings
/s /e
  cscript P:\informatik\smartgit\replaceusername.vbs
)

echo starte smartgit
xcopy H:\smartgit-settings\.gitconfig %USERPROFILE%\ /I /Y
p:\informatik\smartgit\bin\smartgit.exe
xcopy %USERPROFILE%\gitconfig H:\smartgit-settings /Y /I
```

Der erste Teil kopiert das Settingsverzeichnis, wenn es noch nicht existiert. Der Username wird durch ein vbs-Skript ausgelesen und angepasst.

```
Const ForReading = 1
Const ForWriting = 2

strFileName = "H:\smartgit-settings\preferences.yml"
strOldText = "%user%"
strNewText = CreateObject("WScript.Network").UserName

Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile(strFileName, ForReading)
```

```
strText = objFile.ReadAll  
objFile.Close  
  
strNewText = Replace(strText, strOldText, strNewText)  
Set objFile = objFSO.OpenTextFile(strFileName, ForWriting)  
objFile.Write strNewText  
objFile.Close
```

Der zweite Teil ist notwendig, wenn das %USERPROFILE% der Schüler nicht gespeichert wird. In der .gitconfig-Datei werden globale Git-Einstellungen gespeichert. Durch die Befehle wird die in smartgit-settings gesicherte .gitconfig an die richtige Stelle kopiert, dann Smartgit gestartet und eventuelle Änderungen nach Beendigung wieder gesichert. Das ist notwendig, um Benutzer und Email-Konfiguration in Git dauerhaft zu speichern.

1.3.2. Repository eröffnen

Nach der Installation und dem Einstellen der Initialparameter findet sich die Einstellung zum Einbinden eines neuen Repositories oben links.

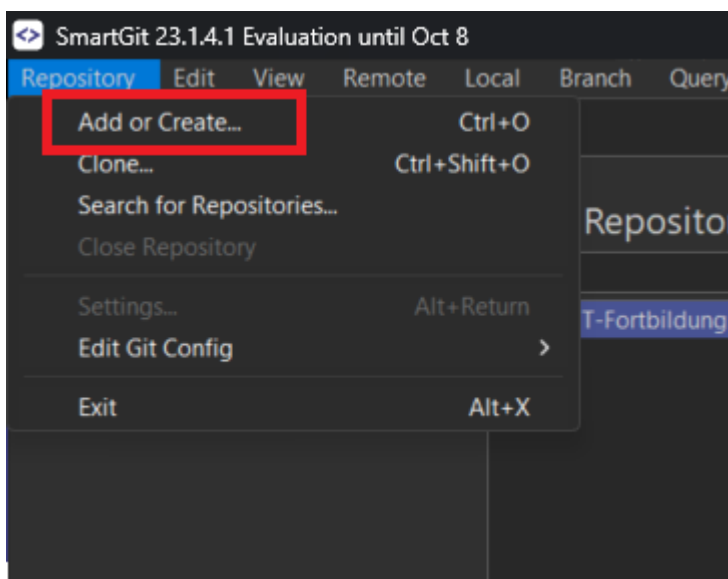


Abb. 6. Neues Repository anlegen oder abrufen

Durch Clicken der Option öffnet sich ein neues Dialogfeld:

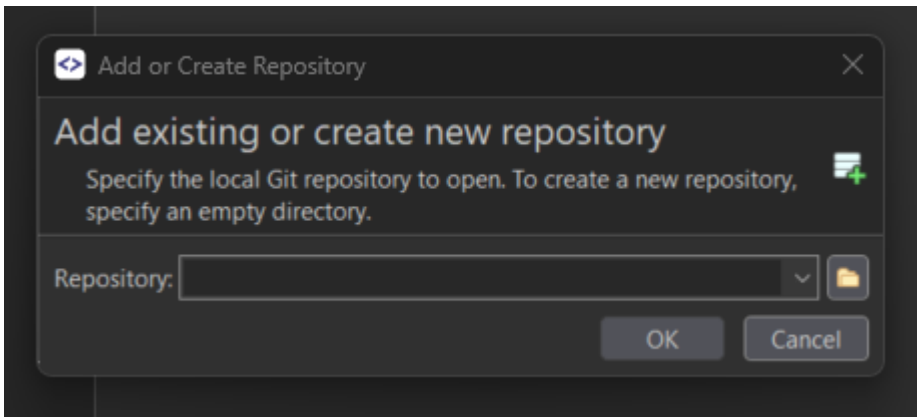


Abb. 7. Dialogfeld Neuanlegung

Wählen Sie hier einen Speicherort für das entsprechende Repository aus. Der Ordner sollte im Regelfall leer sein.

Durch das Auswählen von "Remote" in der Leiste am oberen Rand lässt sich eine Verknüpfung zu einem leeren Repository auf dem Git-Server anlegen

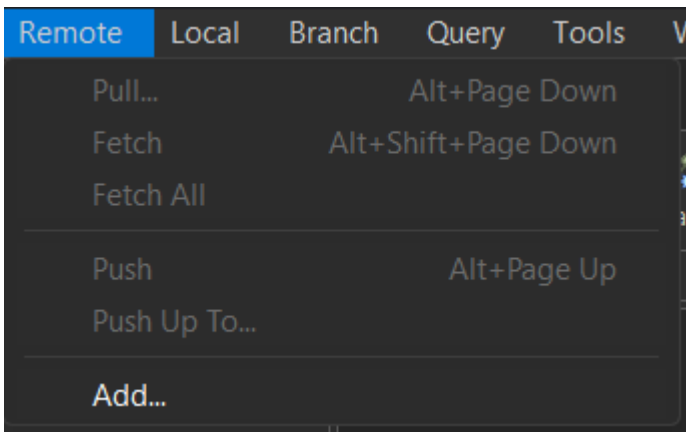


Abb. 8. Repository auf Server auswählen

Nach der Authentifizierung sind nun die beiden Repositories verknüpft und lassen sich fortan vollumfänglich benutzen.

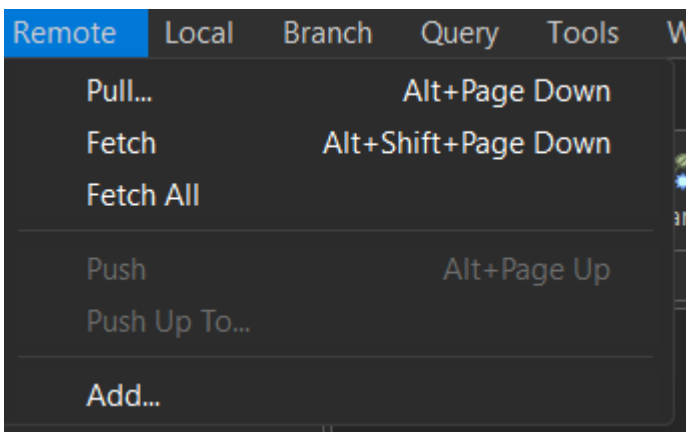


Abb. 9. Pull-Optionen bei vollständiger Verknüpfung

1.3.3. Repository klonen

Möchten Sie ein bestehendes Repository auf Ihren lokalen Rechner übertragen sprechen wir ja bekanntlich vom Klonen. Über Repository → Clone öffnet sich die Dialogmaske zum Eintragen des entsprechenden Links.

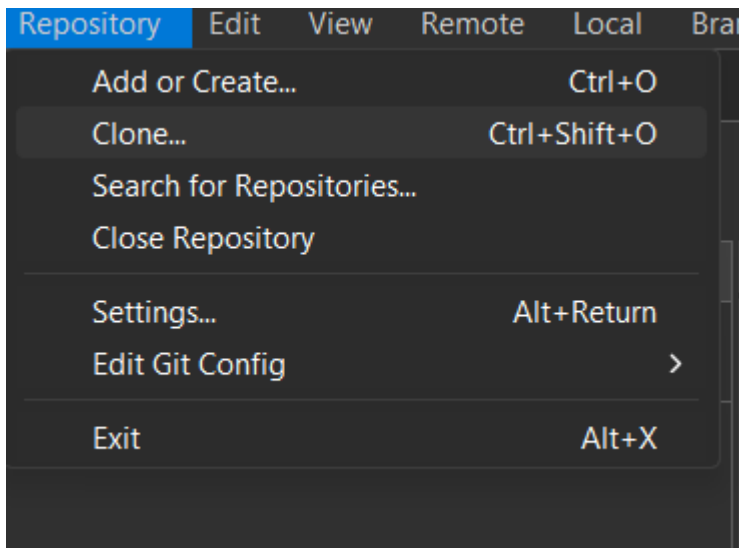


Abb. 10. Auswahl der Clone-Option

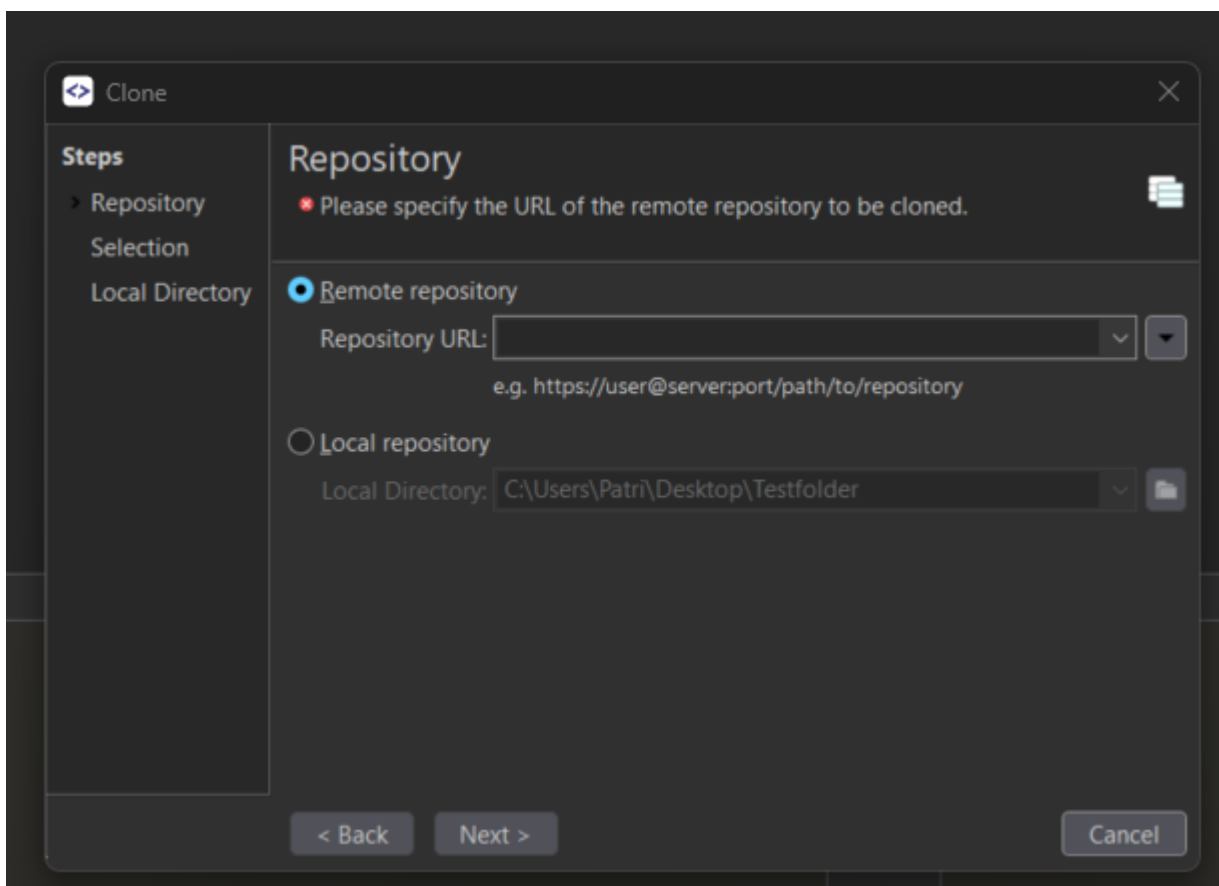


Abb. 11. Clone-Dialog

Da die beiden Optionen selbsterklärend sind wird nicht weiter darauf eingegangen. Die folgenden Dialoge fragen danach nach dem Speicherort und den Klonoptionen. Sobald dies

ausgewählt wurde öffnet sich das Projekt im Betrachter.

1.3.4. Übersichten

Nachdem das entsprechende Repository erfolgreich in SmartGit eingebunden wurde bietet es sich an zwischen den entsprechenden Hauptansichten zu wählen.

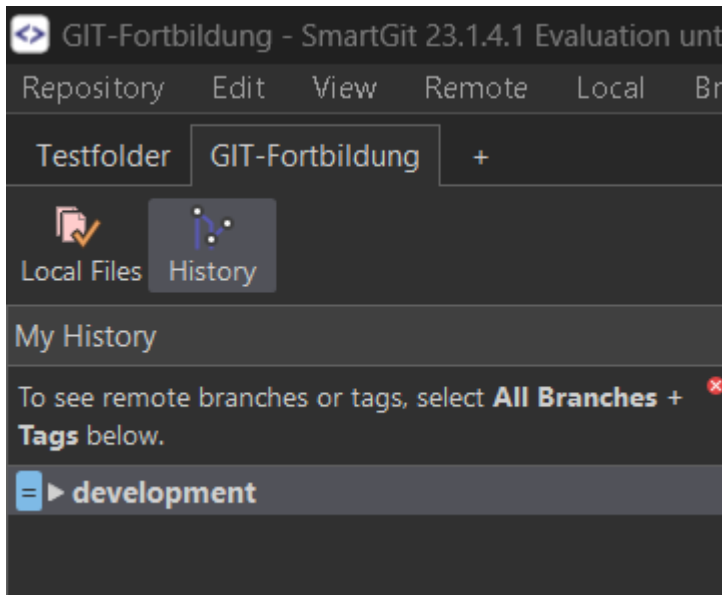


Abb. 12. Hauptansichten

- Local Files dient dabei dazu, die Dateien direkt zu verwalten. Hier sieht man eine Übersicht der Dateien seit dem letzten Pull und kann entscheiden welche Dateien damit im nächsten Commit landen.
- History zeigt den Branchverlauf der letzten Commits und deren entsprechenden Branches. Durch Anklicken der entsprechenden Commits wird ein sog. Diff erstellt, welches die Veränderungen durch den entsprechenden Commit darstellt. Gelöschte Zeilen sind dabei rot, grüne wurden neu hinzugefügt.

In beiden Ansichten finden sich die Branching-Optionen oben in der Mitte:

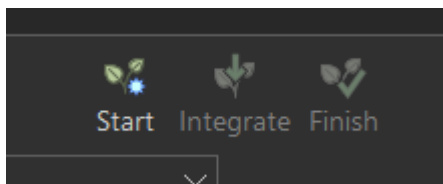


Abb. 13. BranchingOptions

Branches werden hier Feature genannt.

1.3.5. Mergen und Mergekonflikte

Um den Mergevorgang nachzustellen wurde folgendes Szenario erschaffen: Eine Datei wurde von zwei Rechnern gleichzeitig bearbeitet und hinterher zunächst von einem erfolgreich gepusht. Der zweite Rechner versucht nun seine Version ebenfalls zu pushen.

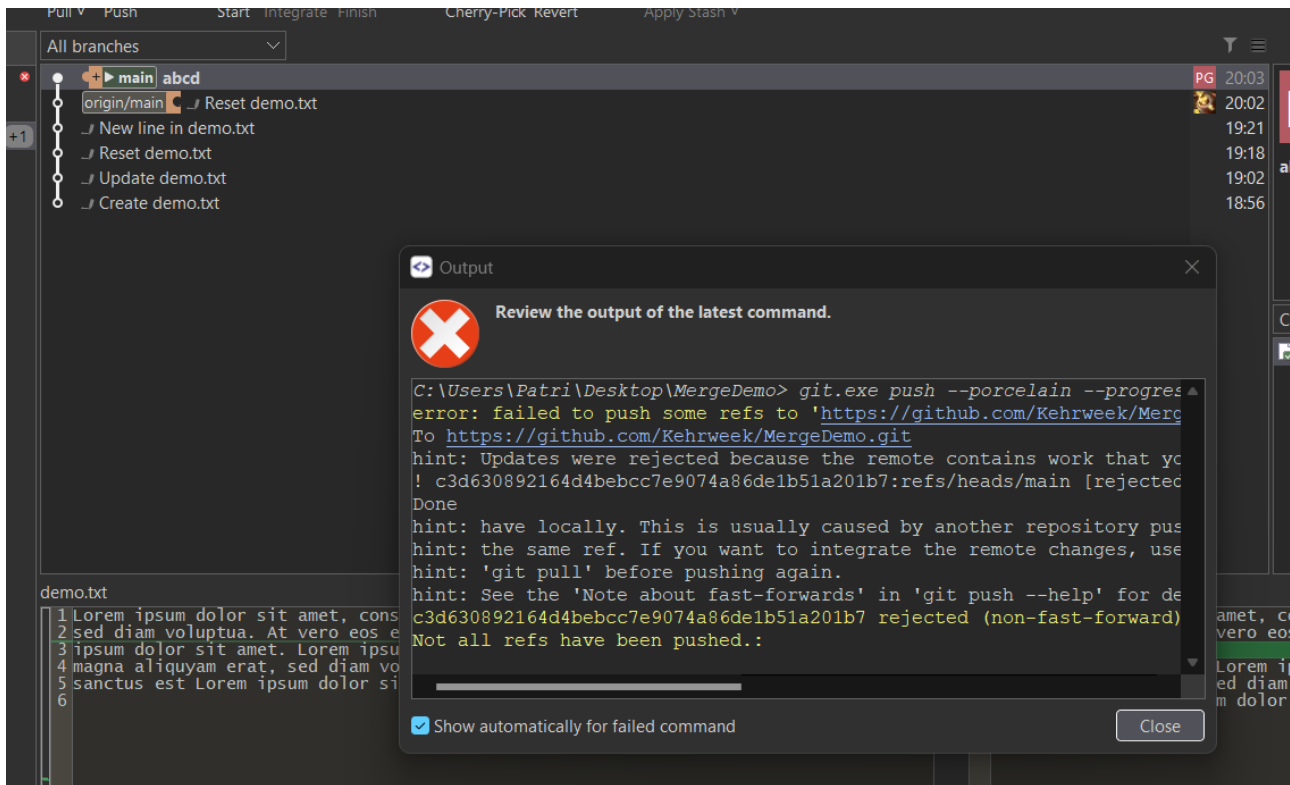


Abb. 14. Smartgit Mergekonflikt

Daraufhin erstellt Smartgit eine Auflistung der Mergekonflikte.

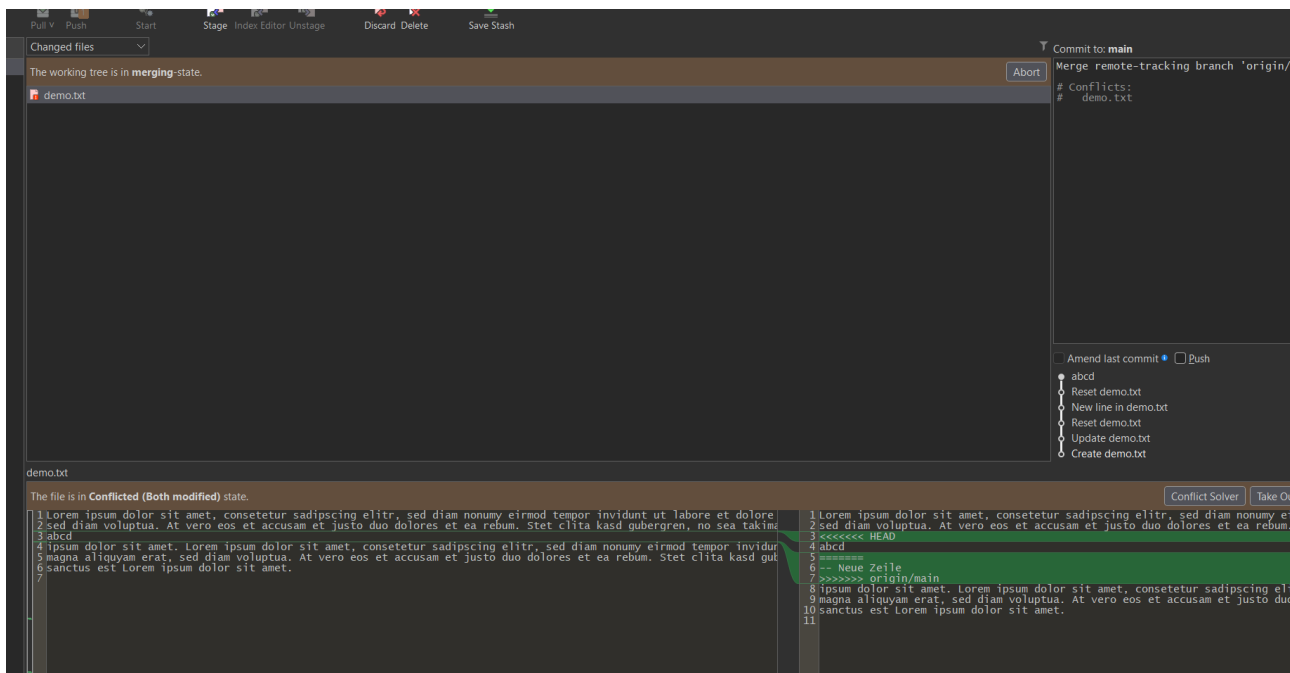


Abb. 15. Smartgit Merging Teil 1

Außerdem werden verschiedene Buttons angeboten um den Konflikt zügig zu lösen.

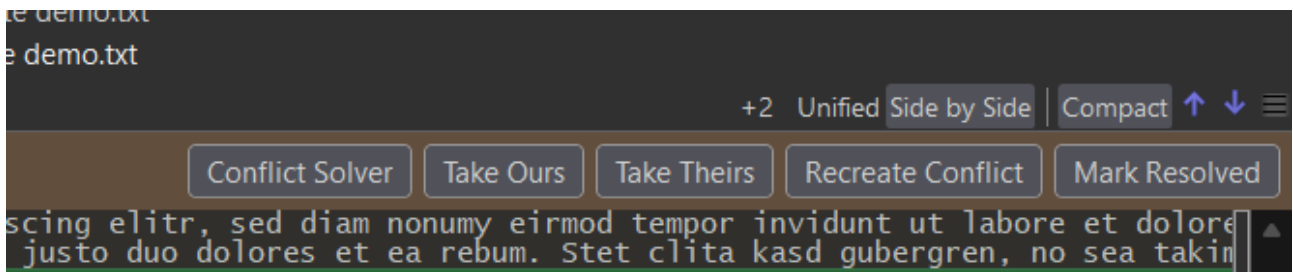


Abb. 16. Smartgit Merging Teil 2

Durch Klicken auf die Datei öffnet sich ein detailliertes Fenster um in dieser jeweils zeilenweise zu entscheiden welcher Teil übernommen werden soll.

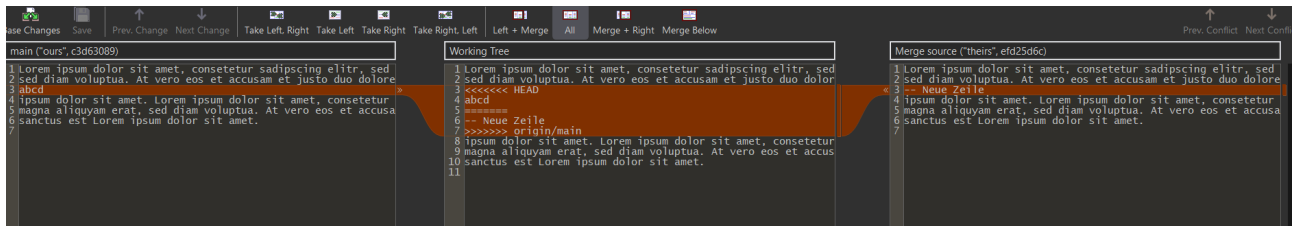


Abb. 17. Smartgit Merging Teil 3

Die Entscheidung wird durch das Klicken auf die kleinen Pfeile getätigt.,

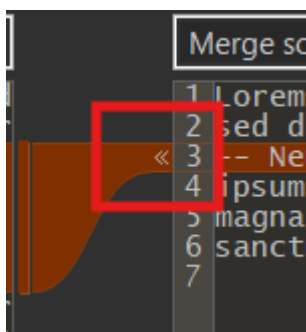


Abb. 18. Smartgit Merging Teil 4

Ist der Konflikt entschieden wird die ignorierte / herausgelöschte Seite rot dargestellt.

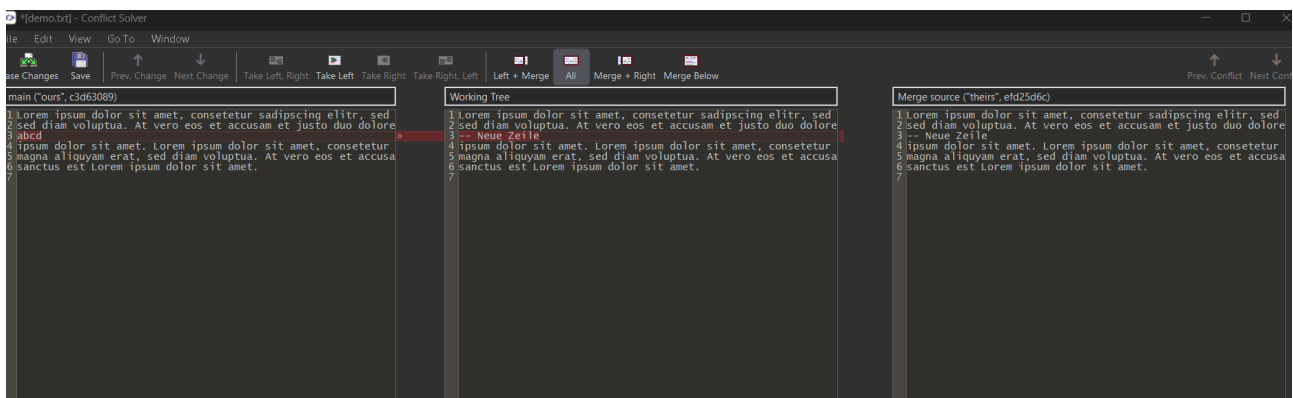


Abb. 19. Smartgit Merging Teil 5

Sind mehrere Konflikte vorhanden kann über diese beiden Pfeile zwischen diesen gewechselt werden.

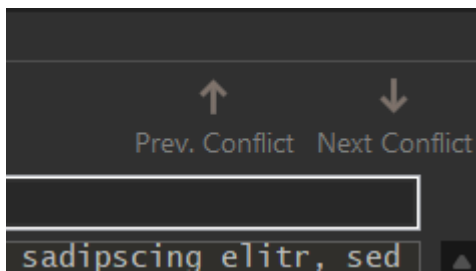


Abb. 20. Smartgit Merging Teil 6

Ist die Konfliktentscheidung abgeschlossen muss die entsprechende Datei als resolved markiert werden.

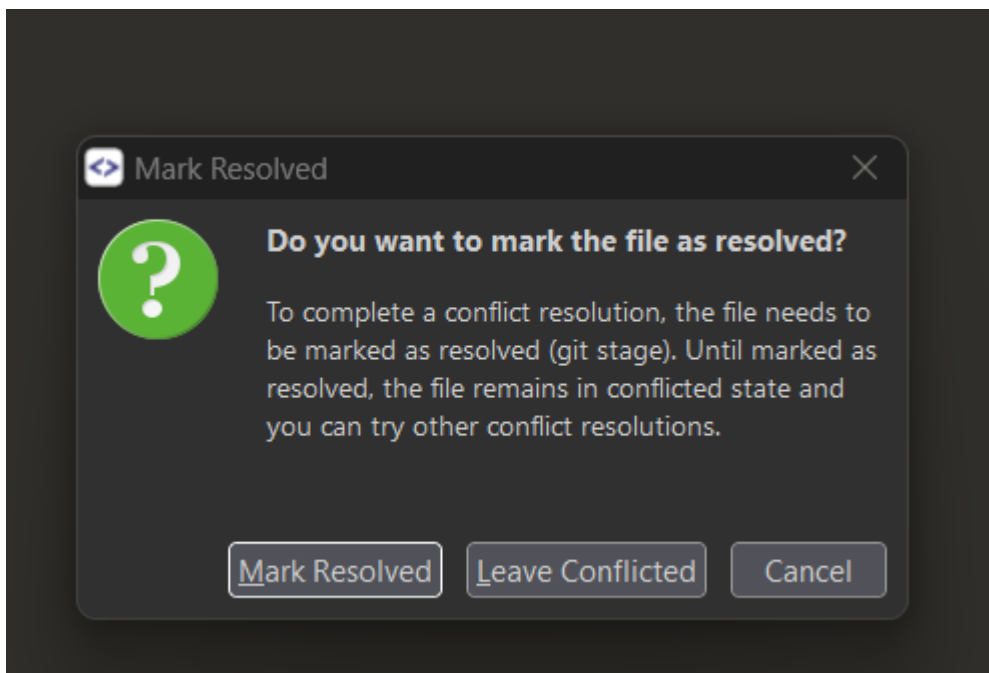


Abb. 21. Smartgit Merging Teil 7

Der gelöste Mergekonflikt wird hinterher in der History durch die Vereinigung der jeweiligen Branches visualisiert.

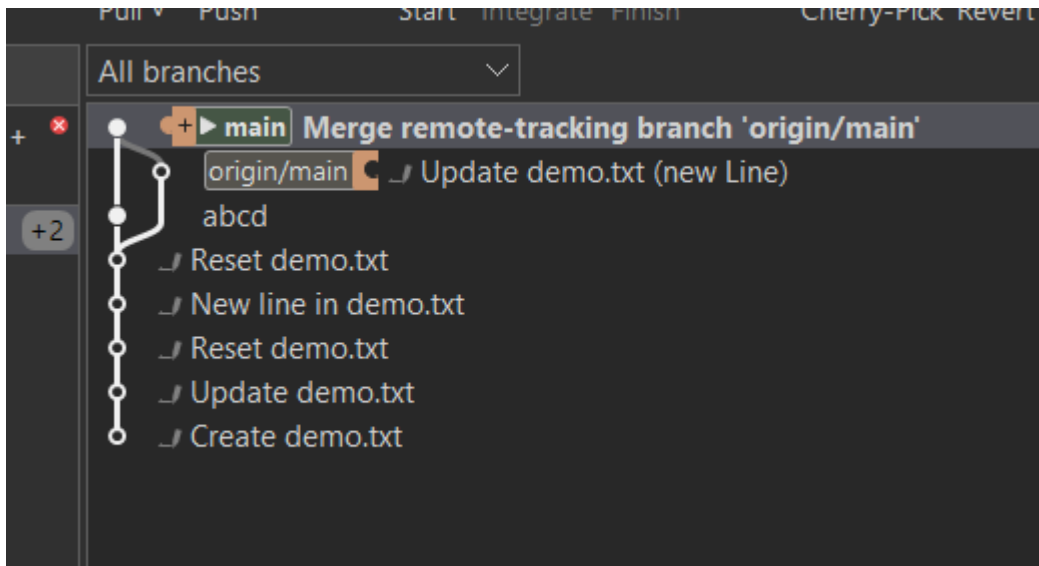


Abb. 22. Smartgit Merging Teil 8

Entwickelt man ohne entsprechende Mergekonflikte werden Branches über Pull-Requests auf der Website vereinigt.

2. Git-Konsole

2.1. Anlegen eines Repositories

Um ein neues Git-Repository mit der Git-Konsole anzulegen, befolgen Sie bitte diese Schritte:

2.1.1. Konsole starten

Starten Sie Ihre bevorzugte Git-Konsole. Dies könnte die Befehlszeile (Command Line) oder eine integrierte Konsole in Ihrer Entwicklungsumgebung sein, wie zum Beispiel Git Bash.

2.1.2. Navigation

Verwende den Befehl `cd` (Change Directory), um zum gewünschten Speicherort zu navigieren, an dem Sie das neue Repository erstellen möchten.

2.1.3. Initialisierung

Verwenden Sie den Befehl `git init`, um ein neues Git-Repository im aktuellen Verzeichnis zu initialisieren. Dieser Befehl legt ein neues leeres Repository an.

2.1.4. Verzeichnis prüfen

Bestätigen Sie, dass das neue Git-Repository erfolgreich initialisiert wurde, indem Sie den Befehl `ls -a` (List All) verwenden, um den Inhalt des aktuellen Verzeichnisses anzuzeigen. Sie sollten ein verstecktes `.git`-Verzeichnis sehen, das das Repository enthält.

2.2. Clonen eines Repositories

Wenn Sie ein vorhandenes Git-Repository auf Ihrem lokalen System klonen möchten, können Sie dies mit der Git-Konsole tun. Hier sind die Schritte dazu:

2.2.1. Konsole starten

Öffnen Sie die Git-Konsole Ihrer Wahl. Sie können die Befehlszeile (Command Line) verwenden oder eine integrierte Konsole in Ihrer Entwicklungsumgebung wie Git Bash oder das integrierte Terminal.

2.2.2. Navigation

Verwenden Sie den Befehl `cd` (Change Directory), um zum Verzeichnis zu navigieren, in dem Sie das geklonte Repository speichern möchten.

2.2.3. Klonvorgang anstoßen

Verwenden Sie den Befehl `git clone` gefolgt von der URL des Repositories, das Sie klonen möchten. Zum Beispiel:

```
git clone https://fortbildner.gitcamp-bw.de/ts-zsl-rska/GIT-Fortbildung.git
```

Hier werden Sie nach Ihren Login-Daten gefragt, welche Sie eingeben müssen, um Zugang zu den Dateien zu erlangen.

2.2.4. Erfolg prüfen

Sobald der Klonvorgang abgeschlossen ist, können Sie mit `ls` den Inhalt des aktuellen Verzeichnisses überprüfen. Sie sollten das geklonte Repository als neues Verzeichnis sehen.

2.3. Stage / Commit

Wenn Sie Änderungen an Ihrem Git-Projekt vorgenommen haben und diese für einen Commit vorbereiten möchten, können Sie dies einfach über die Git-Konsole tun. Hier sind die Schritte dazu:

2.3.1. Überprüfen der Änderungen

Verwenden Sie den Befehl `git status`, um den aktuellen Status Ihres Repositories zu überprüfen. Dadurch erhalten Sie eine Liste der geänderten, ungestageten Dateien.

2.3.2. Stagen der Änderungen

Verwenden Sie den Befehl `git add`, um die gewünschten Änderungen zur Staging-Area

hinzufügen. Sie können einzelne Dateien oder Verzeichnisse hinzufügen, indem Sie ihren Pfad angeben, oder alle Änderungen auf einmal mit einem Punkt (.) für alle Dateien im aktuellen Verzeichnis.

Beispiel für das Stagen einer einzelnen Datei:

```
git add Dateiname
```

Beispiel für das Stagen aller Änderungen im aktuellen Verzeichnis:

```
git add .
```

2.3.3. Überprüfen der gestagten Änderungen

Verwenden Sie erneut den Befehl `git status`, um sicherzustellen, dass die gewünschten Änderungen erfolgreich zur Staging-Area hinzugefügt wurden. Sie sollten eine Liste der gestagten Änderungen sehen.

2.3.4. Comitten der Änderungen

Verwenden Sie den Befehl `git commit`, um die gestagten Änderungen zu committen. Geben Sie eine aussagekräftige Commit-Nachricht ein, um die durchgeführten Änderungen zu beschreiben.

```
git commit -m "Hier ist Ihre Commit-Nachricht"
```

2.3.5. Überprüfen des Commit-Erfolgs

Nachdem Sie die Änderungen committet haben, können Sie mit `git log` den Commit-Verlauf anzeigen und sicherstellen, dass Ihr Commit erfolgreich war.

2.4. Check out

Wenn Sie an einem bestimmten Branch oder einem früheren Commit in Ihrem Git-Repository arbeiten möchten, können Sie dies über die Git-Konsole tun. Hier sind die Schritte dazu:

2.4.1. Überprüfen des Repository-Status

Verwenden Sie den Befehl `git status`, um den aktuellen Status Ihres Repositories zu überprüfen. Dadurch erhalten Sie Informationen darüber, ob Sie Änderungen haben, die committet oder gestaged werden müssen.

2.4.2. Aus-checken des Gewünschten Branchs oder Commits

Verwenden Sie den Befehl `git checkout`, um zum gewünschten Branch oder Commit zu wechseln. Geben Sie den Namen des Branchs oder die Commit-ID an, zu der Sie wechseln möchten.

Beispiel für das Auschecken eines Branchs:

```
git checkout Branch-Name
```

Beispiel für das Auschecken eines früheren Commits:

```
git checkout Commit-ID
```

2.4.3. Überprüfen des Wechsel-Erfolgs

Nachdem Sie zum gewünschten Branch oder Commit gewechselt haben, verwenden Sie `git status`, um sicherzustellen, dass der Wechsel erfolgreich war und Ihr Arbeitsverzeichnis auf dem neuen Stand ist.

2.5. Push

Wenn Sie Ihre lokalen Änderungen an ein entferntes Git-Repository hochladen möchten, können Sie dies über die Git-Konsole tun. Hier sind die Schritte dazu:

2.5.1. Überprüfen des Repository-Status

Verwenden Sie den Befehl `git status`, um den aktuellen Status Ihres Repositories zu überprüfen. Dadurch erhalten Sie Informationen darüber, ob Sie Änderungen haben, die committet oder gestaged werden müssen.

2.5.2. Pushen der Änderungen

Verwenden Sie den Befehl `git push`, um Ihre lokalen Änderungen auf das entfernte Repository hochzuladen. Geben Sie den Namen des entfernten Repositories und den Namen des Branchs an, auf den Sie pushen möchten.

Beispiel für das Pushen auf den Hauptbranch (üblicherweise "master" oder "main"):

```
git push origin master
```

2.5.3. Authentifizierung (falls erforderlich)

Je nach Konfiguration des entfernten Repositories kann es sein, dass Sie sich authentifizieren müssen, um den Push-Vorgang abzuschließen. Geben Sie Ihre Anmeldeinformationen ein, wenn Sie dazu aufgefordert werden.

2.5.4. Überprüfen des Push-Erfolgs

Nachdem der Push-Vorgang abgeschlossen ist, können Sie die Repository-Website besuchen oder den Befehl `git log` verwenden, um sicherzustellen, dass Ihre Änderungen erfolgreich auf das entfernte Repository hochgeladen wurden.

2.6. Pull

Wenn Sie die neuesten Änderungen aus einem entfernten Git-Repository auf Ihr lokales Repository herunterladen möchten, können Sie dies über die Git-Konsole tun. Hier sind die Schritte dazu:

2.6.1. Pullen der neuesten Änderungen

Verwenden Sie den Befehl `git pull`, um die neuesten Änderungen aus dem entfernten Repository herunterzuladen und mit Ihrem lokalen Repository zu fusionieren. Geben Sie den Namen des entfernten Repositories und den Namen des Branchs an, den Sie pullen möchten.

Beispiel für das Pullen aus dem Hauptbranch (üblicherweise "master" oder "main"):

```
git pull origin master
```

2.6.2. Authentifizierung (falls erforderlich)

Je nach Konfiguration des entfernten Repositories kann es sein, dass Sie sich authentifizieren müssen, um den Pull-Vorgang abzuschließen. Geben Sie Ihre Anmeldeinformationen ein, wenn Sie dazu aufgefordert werden.

2.6.3. Überprüfen des Pull-Erfolgs

Nachdem der Pull-Vorgang abgeschlossen ist, verwenden Sie den Befehl `git log` oder andere Git-Befehle, um sicherzustellen, dass die neuesten Änderungen erfolgreich in Ihr lokales Repository gezogen wurden.

2.7. Merge / Rebase

Wenn Sie Änderungen aus einem anderen Branch in Ihren aktuellen Branch integrieren möchten, können Sie dies über die Git-Konsole tun. Hier sind die Schritte dazu:

2.7.1. Navigieren zum Ziel-Branch

Verwenden Sie den Befehl `git checkout`, um zum Branch zu wechseln, in den Sie die Änderungen integrieren möchten. Stellen Sie sicher, dass Sie in dem Branch sind, in den Sie die Änderungen mergen möchten.

Beispiel für das Wechseln zum Ziel-Branch:

```
git checkout Ziel-Branch-Name
```

2.7.2. Mergen des Quell-Branchs

Verwenden Sie den Befehl `git merge`, um den Quell-Branch in den Ziel-Branch zu mergen. Geben Sie den Namen des Quell-Branchs an, den Sie mergen möchten.

Beispiel für das Mergen des Quell-Branchs:

```
git merge Quell-Branch-Name
```

2.7.3. Bearbeiten von Merge-Konflikten (falls erforderlich)

Wenn es Merge-Konflikte gibt, die nicht automatisch gelöst werden können, müssen Sie diese manuell bearbeiten. Öffnen Sie die betroffenen Dateien in einem Texteditor, beheben Sie die Konflikte und führen Sie dann den Merge-Vorgang erneut aus. Dieser Schritt wird durch moderne Entwicklungsumgebungen stark vereinfacht.

2.8. Branches

Branches dienen dazu einzelne Features bzw. Teilprojekte separat anzulegen ohne dabei die Integrität des Hauptprojektes zu gefährden. In einer kollaborativen Entwicklungssituation sind sie unabdingbar.

2.8.1. anlegen

Sollten Sie einen bisherigen branch um einen neuen erweitern wollen, so sind folgende Schritte zu beachten:

Überprüfen des Repository-Status

Verwenden Sie den Befehl `git status`, um den aktuellen Status Ihres Repositories zu überprüfen. Dadurch erhalten Sie Informationen darüber, ob Sie ungespeicherte Änderungen haben, die vor dem Branches committet oder gestaged werden müssen.

Anlegen des Neuen Branchs

Verwenden Sie den Befehl `git branch`, um einen neuen Branch anzulegen. Geben Sie den Namen des neuen Branchs an, den Sie erstellen möchten.

Beispiel für das Anlegen eines neuen Branchs:

```
git branch Neuer-Branch-Name
```

Wechseln zum Neuen Branch

Verwenden Sie den Befehl `git checkout`, um zum neu erstellten Branch zu wechseln und dort zu arbeiten.

Beispiel für das Wechseln zum neuen Branch:

```
git checkout Neuer-Branch-Name
```

Überprüfen des Branch-Erfolgs

Nachdem Sie den neuen Branch erstellt haben, verwenden Sie den Befehl `git branch` erneut oder andere Git-Befehle, um sicherzustellen, dass der neue Branch erfolgreich erstellt wurde und Sie sich in ihm befinden.

2.8.2. mergen

Wenn Sie die Änderungen aus einem anderen Branch in Ihren aktuellen Branch integrieren möchten, können Sie dies über die Git-Konsole tun. Hier sind die Schritte dazu:

Überprüfen des Repository-Status

Verwenden Sie den Befehl `git status`, um den aktuellen Status Ihres Repositorys zu überprüfen. Dadurch erhalten Sie Informationen darüber, ob Sie ungespeicherte Änderungen haben, die vor dem Mergen committet oder gestaged werden müssen.

Wechseln zum Ziel-Branch

Verwenden Sie den Befehl `git checkout`, um zum Branch zu wechseln, in den Sie die Änderungen mergen möchten. Stellen Sie sicher, dass Sie sich im Ziel-Branch befinden, in den Sie die Änderungen integrieren möchten.

Beispiel für das Wechseln zum Ziel-Branch:

```
git checkout Ziel-Branch-Name
```

Mergen des Quell-Branchs

Verwenden Sie den Befehl `git merge`, um den Quell-Branch in den Ziel-Branch zu mergen. Geben Sie den Namen des Quell-Branchs an, den Sie mergen möchten.

Beispiel für das Mergen des Quell-Branchs:

```
git merge Quell-Branch-Name
```

Bearbeiten von Merge-Konflikten (falls erforderlich)

Wenn es Merge-Konflikte gibt, die nicht automatisch gelöst werden können, müssen Sie diese manuell bearbeiten. Öffnen Sie die betroffenen Dateien in einem Texteditor oder Entwicklungsumgebung, beheben Sie die Konflikte und führen Sie dann den Merge-Vorgang erneut aus.

Überprüfen des Merge-Erfolgs

Nachdem der Merge-Vorgang abgeschlossen ist, verwenden Sie den Befehl `git status` oder andere Git-Befehle, um sicherzustellen, dass der Merge erfolgreich war und keine Konflikte mehr vorhanden sind.