

Software Entwicklung & Programmierung

Einführung in Versionierungssoftware

Bilder und Texte der Veranstaltungsfolien und -unterlagen sowie das gesprochene Wort innerhalb der Veranstaltung und Lehr-Lern-Videos dienen allein dem Selbst- bzw. Gruppenstudium. Jede weiterführende Nutzung ist den Teilnehmenden der Moodle-Kurse untersagt, z.B. Verbreitung an andere Studierende, in sozialen Netzwerken, dem Internet!

Darüber hinaus ist ein studentischer Mitschnitt von Webkonferenzen im Rahmen der Lehre nicht erlaubt.

Am Ende dieser Präsenzstunde könnt Ihr:

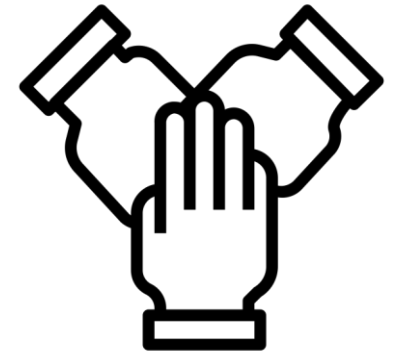
- Typische **Probleme des kollaborativen Arbeitens erläutern** und erklären, wieso diese die Nutzung eines Versionskontrollsystems (wie bspw. Git) erfordern
- Die grundlegende **Arbeitsweise** mit **Git** erläutern
- Das GitLab der Uni-DUE verwenden, um **Dateien** und insbesondere Quellcode zu **versionieren** und in Ihrer **Gruppe gemeinsam zu bearbeiten**

Agenda

1. **Motivation**
2. Git – Grundlagen
3. Git – typischer Ablauf
4. Git – typische Konflikte
5. GitLab der Universität
Duisburg-Essen



- **Kollaboratives Arbeiten** an gemeinsamen Dateien erfordert eine Zugriffskontrolle, da sonst zahlreiche Probleme auftreten können
- Grundlage einer Zugriffskontrolle ist eine gemeinsame **zentrale Stelle** wo alle Dateien die gemeinsam bearbeitet werden abgelegt werden (sogenanntes **Repository**)
- Prinzip ist, dass man jeweils auf einer sogenannten **Arbeitskopie** (oder lokalen Kopie) des Repositories arbeitet
- Hierzu erfolgt zunächst ein **Checkout** aus dem Repository
- Nach der erfolgten Bearbeitung erfolgt ein **Checkin** in das Repository
- Eine **Zugriffskontrolle** fügt die Änderungen systematisch in die Dateien des zentralen Repositories ein und macht sie allen Teammitgliedern zugänglich
- Somit hat jedes Teammitglied immer **Zugriff auf den aktuellen Stand der gemeinsamen Dateien**



- Ein bekanntes und sehr nützliches Werkzeug, welches eine Zugriffskontrolle und Versionsverwaltung bietet, ist Git
- Die Universität bietet hiervon eine kostenlose Version für jeden Studierenden an (Zugang via <https://git.uni-due.de/> und Uni-Kennung)
- Hierzu haben wir euch bereits ein Repository eingerichtet welches als eure gemeinsame zentrale Stelle dient, in der ihr eure Dateien ablegen und verwalten könnt

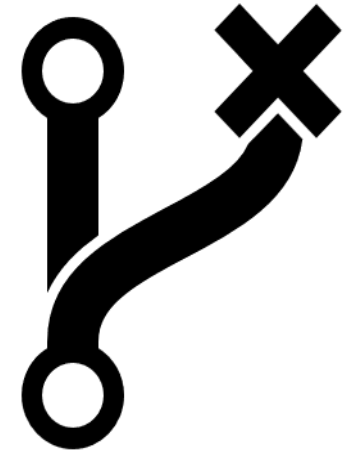
The screenshot shows the login interface for the Gitlab Server of the University of Duisburg-Essen. On the left, there is a header with the university's logo and name, followed by the tagline 'Offen im Denken'. Below this, it states 'Ein Dienst des ZIM (Zentrum für Informations- und Mediendienste)'. A list of instructions for login is provided, including the use of LDAP for Uni-IDs and the Standard tab for external users. At the bottom left, there are links for 'Impressum' and 'Kontakt: git@uni-due.de'. On the right, there is a login form with two tabs: 'LDAP' (selected) and 'Standard'. The form contains fields for 'LDAP Username' and 'Password', a 'Remember me' checkbox, and a green 'Sign in' button.

Agenda

1. Motivation
2. **Git – Grundlagen**
3. Git – typischer Ablauf
4. Git – typische Konflikte
5. GitLab der Universität
Duisburg-Essen

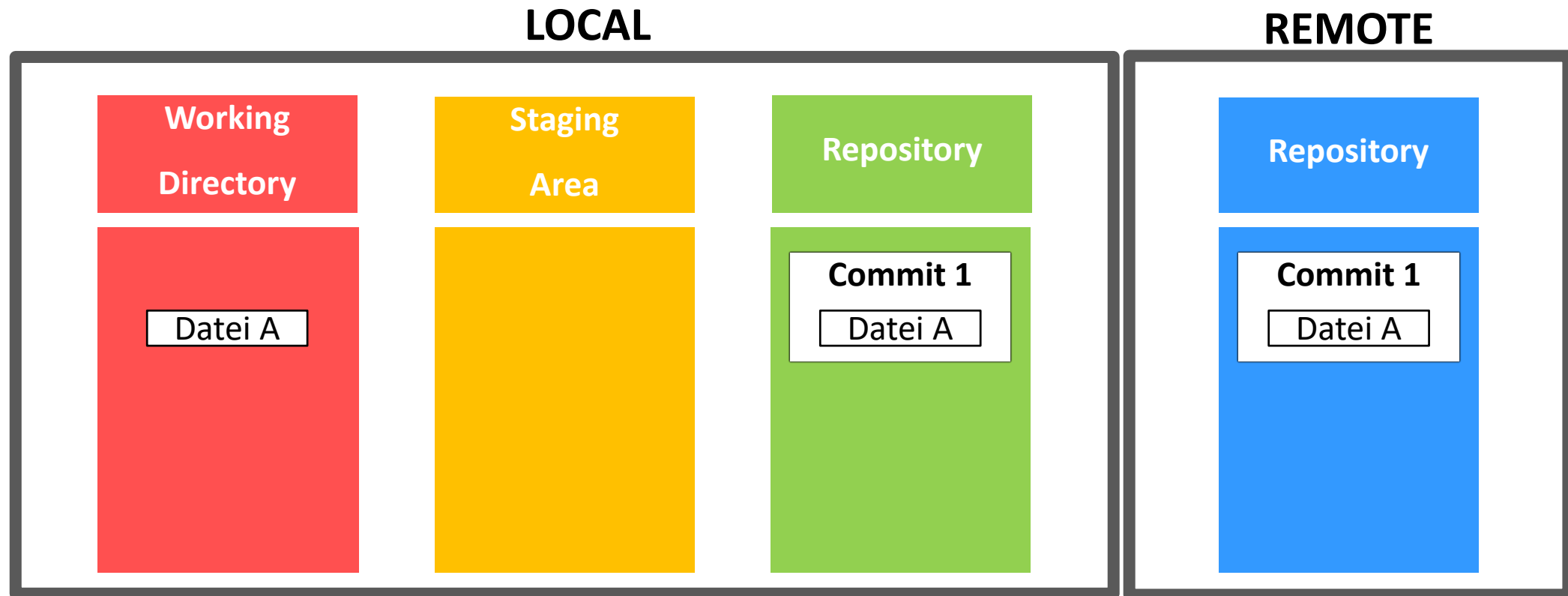


- Erste Open-Source Versionierungssoftware
- Verwaltet gemeinsame Dateibasis (**Repository**)
- Verwaltet verschiedene Versionen dieser Basis
- Versionen können abgezweigt und zusammengeführt werden.
(**Branching, Merging**)



Versionierung in 4 Phasen

- Änderungen werden zu Paketen (Commits) zusammengefasst
- Änderungen durchlaufen folgende 4 Bereiche:



Working Directory

- Umfasst alle lokalen Dateien
- Inklusive Änderungen
- Alle Dateien des Gits befinden sich in einem festgelegten Ordner

Befehle:

- **clone** Initialer Download
- **pull** Download und Einarbeitung aller Commits
- **reset** Zurücksetzen von Änderungen

- Änderungen müssen zuerst manuell hinzugefügt werden
- Hinzugefügte Änderungen werden in einem „Commit“ gespeichert
- Im Regelfall liegen Änderungen nur als Differenz vor → Weniger Datenvolumen

Befehle:

- **add** Fügt Änderungen dem aktuellen Commit hinzu
- **mv** Verschieben ohne Verlust des Änderungsprotokolls

- Umfasst das gesamte lokale Änderungsprotokoll
- Erlaubt lokales Testen
- Unterscheidet sich vom Online (Remote) Repository

Befehle:

- **commit** Hinzufügen des aktuellen Commits
- **fetch** Download aller Änderungen

Online (Remote) Repository

- Umfasst alle Dateien
- Enthält das gesamte hochgeladene Änderungsprotokoll
- Änderungen, die nicht hochgeladen wurden, liegen hier nicht vor!

Befehle

- **push** Hochladen aller lokalen Commits

Agenda

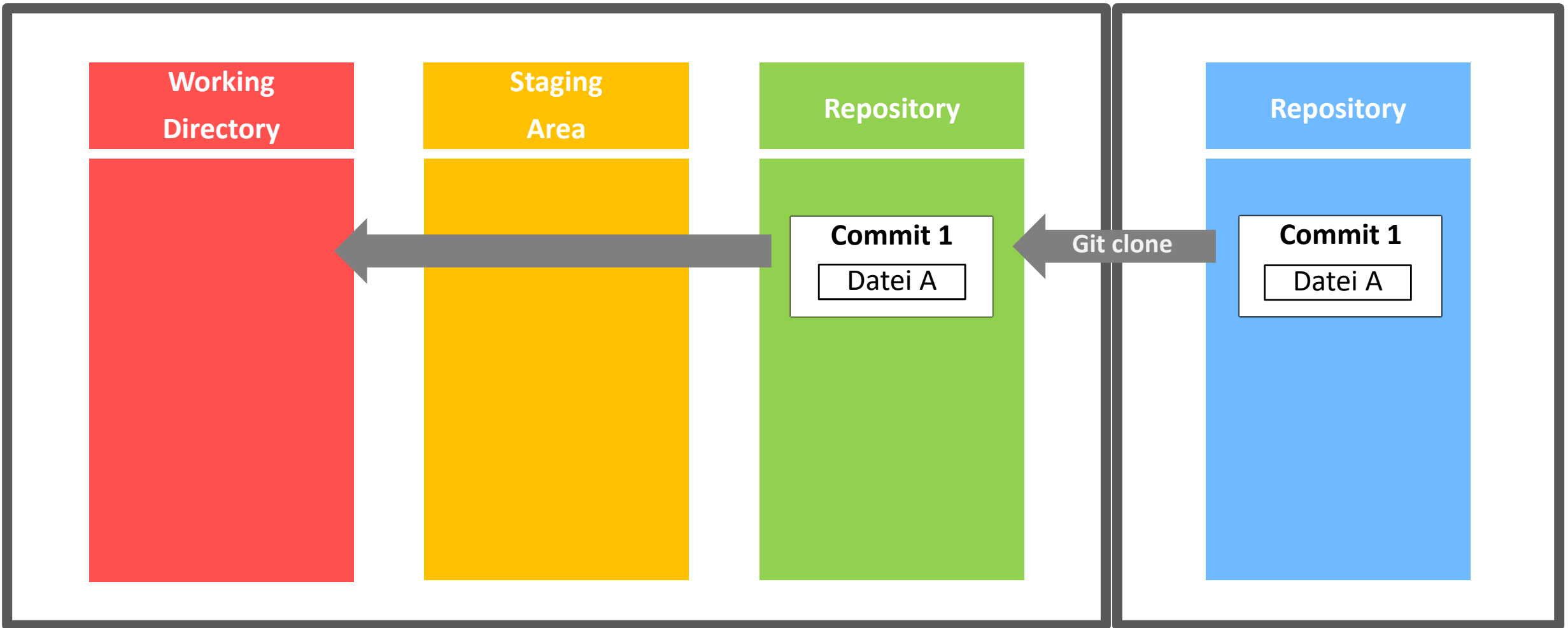
1. Motivation
2. Git – Grundlagen
3. **Git – typischer Ablauf**
4. Git – typische Konflikte
5. GitLab der Universität Duisburg-Essen



Ablauf - clone

LOCAL

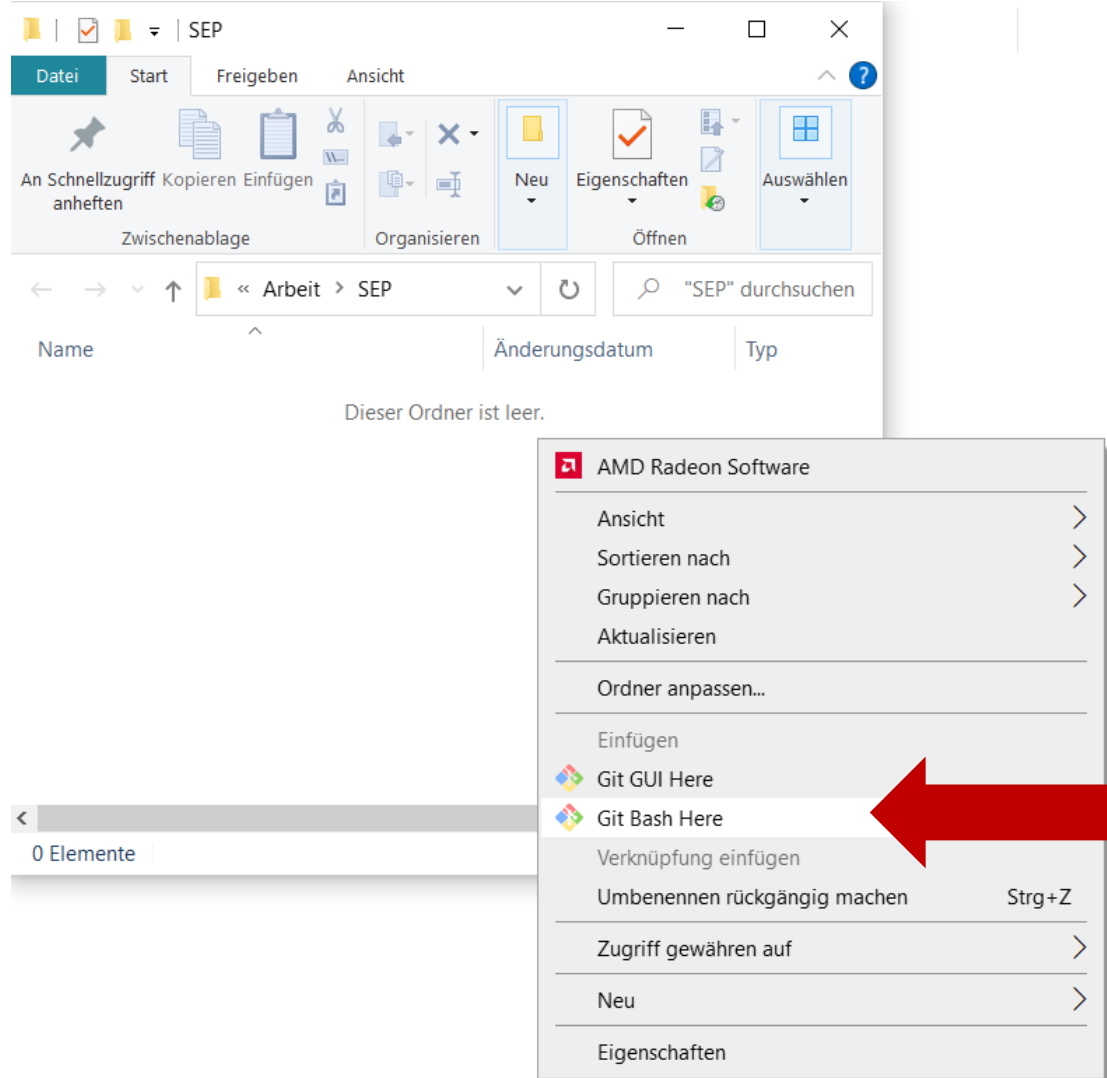
REMOTE



Ablauf - clone

Git Bash

- Wurde **Git Bash** installiert, besteht die Möglichkeit, über einen Rechtsklick die Konsole in einem beliebigen Verzeichnis zu öffnen
- Die Eingabe **git clone <URL-Adresse>** wird mittels des Tastendrucks *Enter* ausgeführt

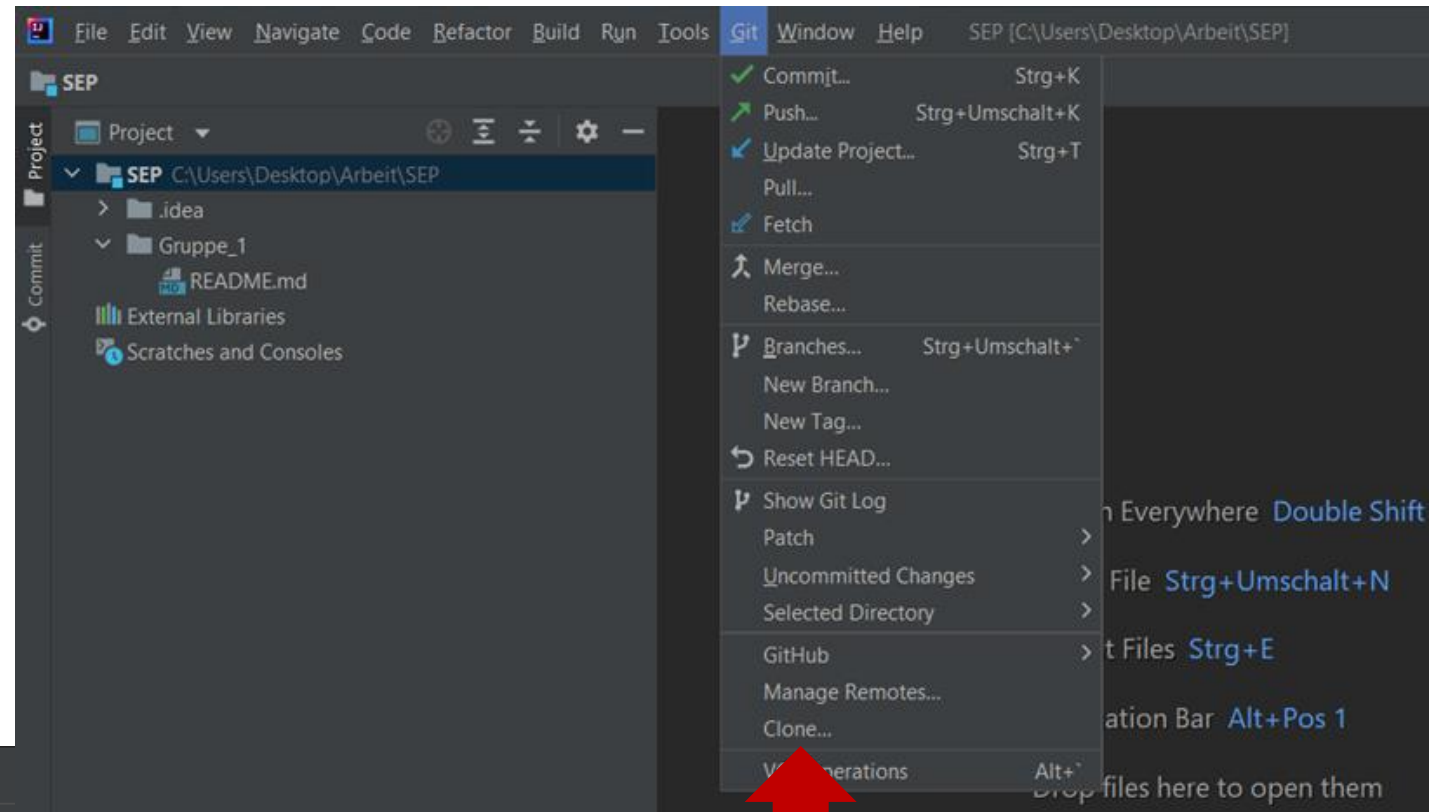
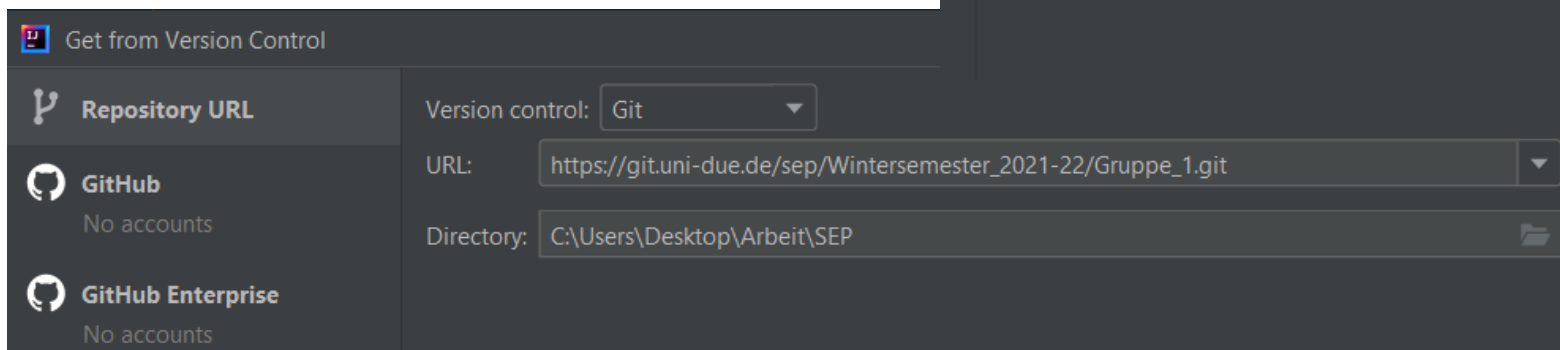


```
MINGW64:/c/Users/Desktop/Arbeit/SEP
$ git clone https://git.uni-due.de/sep/wintersemester_2021-22/Gruppe_1.git
Cloning into 'Gruppe_1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

Ablauf - clone

IntelliJ

- Über den Reiter *Git* öffnet sich ein Drop-Down Menü mit der Option *Clone*
- In dem neuen Fenster wird die **URL** des Repositorys eingetragen (optional kann zusätzlich der Pfad geändert werden)
- Der Vorgang wird mit der Bestätigung über den *Clone*-Button abgeschlossen

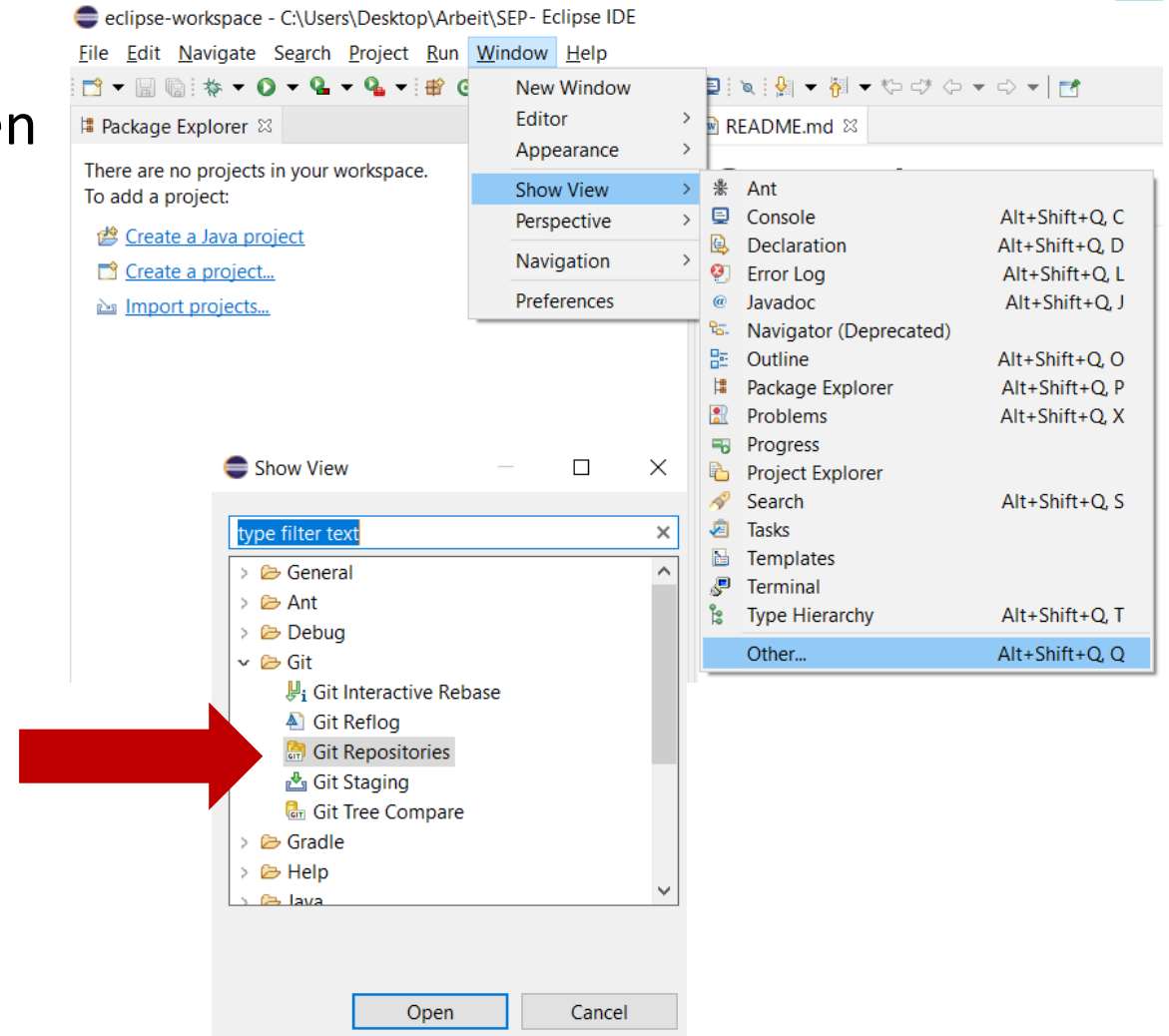


Ablauf - clone

Eclipse

Wird das Git-Fenster in Eclipse beim Starten nicht angezeigt, müssen folgende Schritte durchlaufen werden:

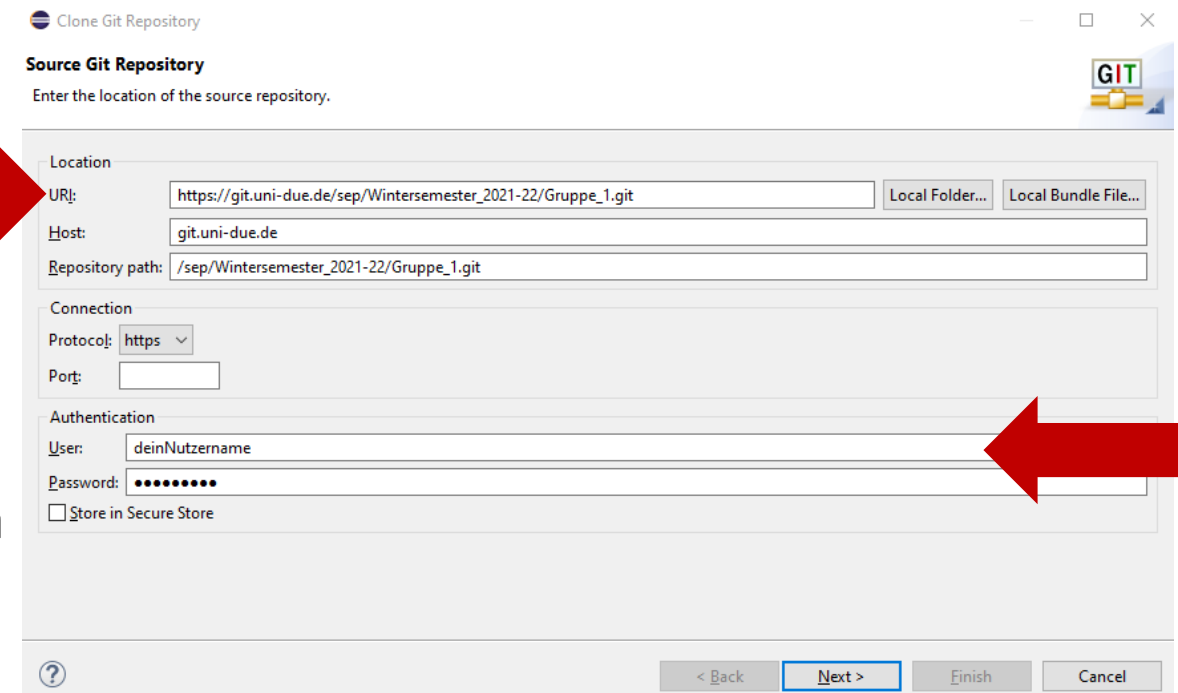
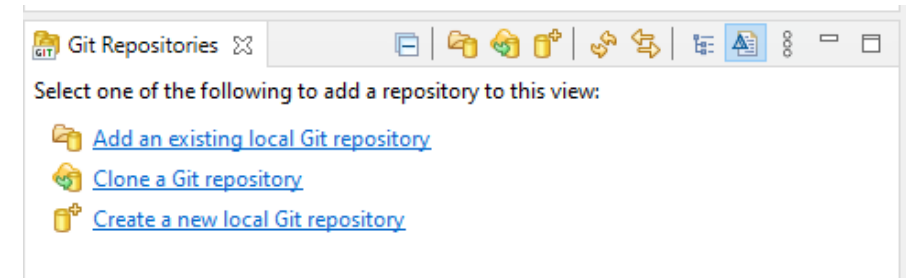
1. Über den Reiter *Window* öffnet sich ein Drop-Down Menü mit der Option *Show View*
2. Anschließend wird der Punkt *Other* ausgewählt
3. In dem neuen Fenster den Ordner **Git** suchen, in diesem *Git Repositories* selektieren und öffnen



Ablauf - clone

Eclipse

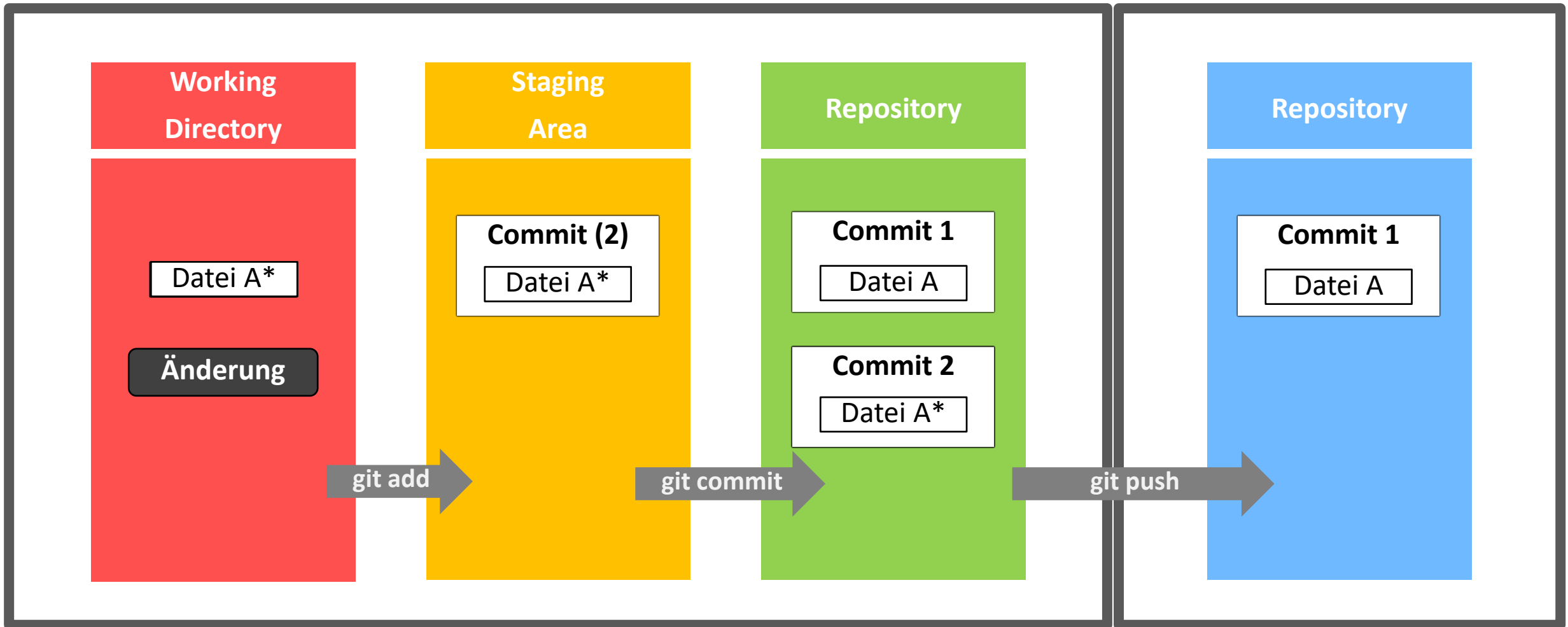
- Im Git-Fenster kann nun die Option *Clone a Git Repository* ausgewählt werden
- Hier nun die **URL** des Git-Repositorys eingeben, sowie die **Uni-Kennung** und das **Passwort**
- In den darauf folgenden Fenstern kann optional der Pfad angepasst werden



Ablauf - Änderungen veröffentlichen

LOCAL

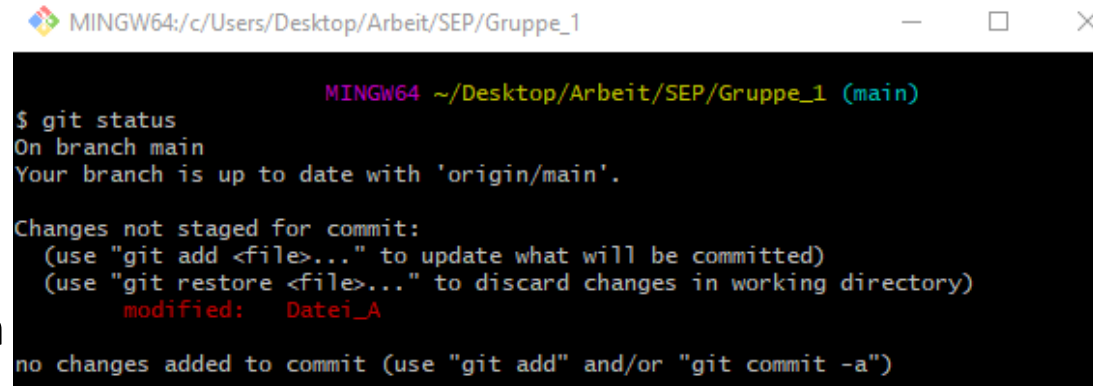
REMOTE



Ablauf - Änderungen veröffentlichen

Git Bash

- Git Bash erkennt automatisch die Dateien, an denen Veränderungen vorgenommen wurden. Diese können über den Befehl **git status** angezeigt werden.
- Mit dem Befehl **git add .** können alle vorgenommenen Änderungen für den nächsten Commit vorgemerkt werden (Diese befinden sich anschließend in der **Staging Area**)
- Mit **git add <Dateiname>** lassen sich ausgewählte Dateien für den nächsten Commit vormerken
- **Rot markierte** Dateien wurden verändert, jedoch nicht für einen Commit vorgemerkt
- **Grün markierte** Dateien sind für den nächsten Commit vorgemerkt

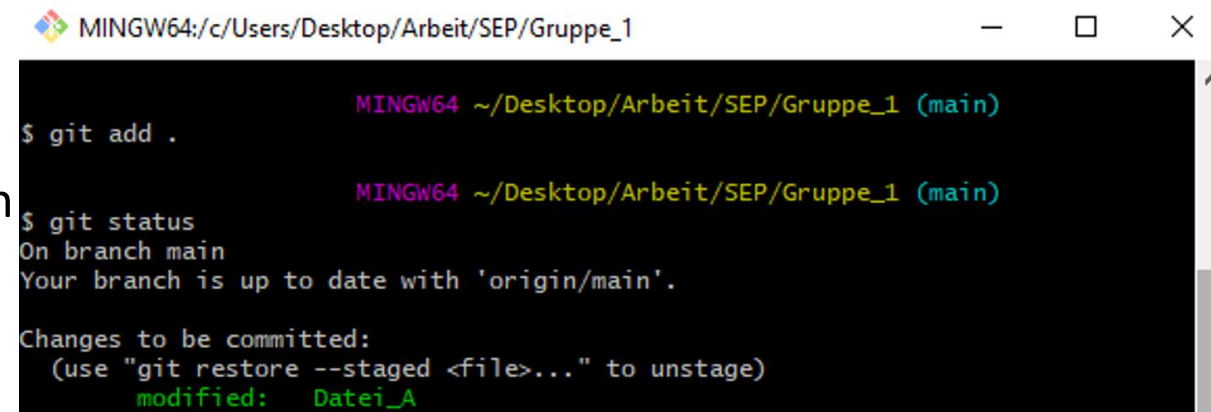


```
MINGW64:/c/Users/Desktop/Arbeit/SEP/Gruppe_1

MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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:   Datei_A

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



```
MINGW64:/c/Users/Desktop/Arbeit/SEP/Gruppe_1

MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)
$ git add .

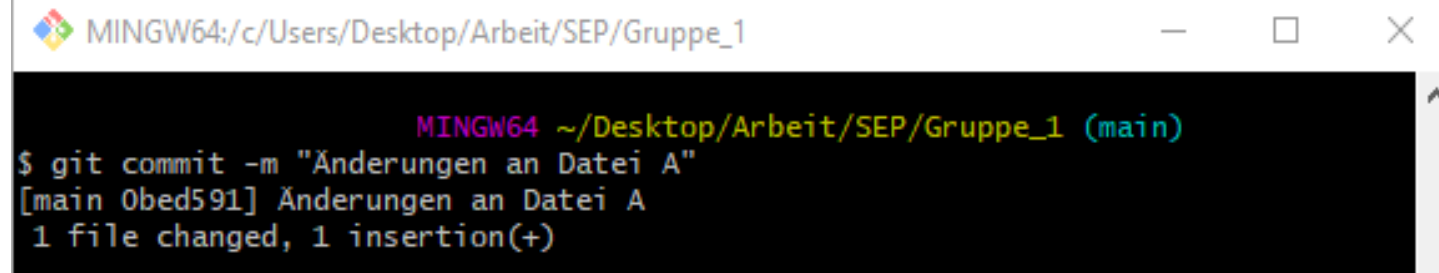
MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Datei_A
```

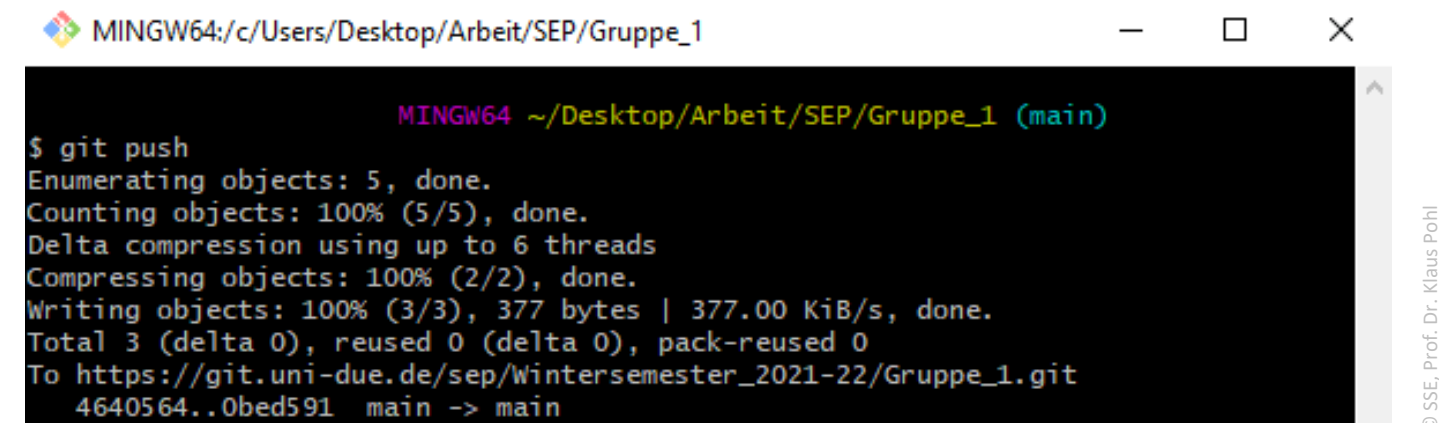
Ablauf - Änderungen veröffentlichen

Git Bash

- Um die vorgemerkten Änderungen zu committen wird der Befehl **git commit -m „Eine Nachricht“** benötigt
- „Eine Nachricht“ dient hier als Platzhalter, dieser Bereich kann genutzt werden, um die vorgenommenen Änderungen zu dokumentieren (Die Nachricht wird **Commit-Message** genannt und ist nach dem Veröffentlichen für andere einsehbar)
- Der angefertigte Commit wird mit dem Befehl **git push** und nach Eingabe des **Passworts** veröffentlicht



```
MINGW64:/c/Users/Desktop/Arbeit/SEP/Gruppe_1
MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)
$ git commit -m "Änderungen an Datei A"
[main 0bed591] Änderungen an Datei A
1 file changed, 1 insertion(+)
```

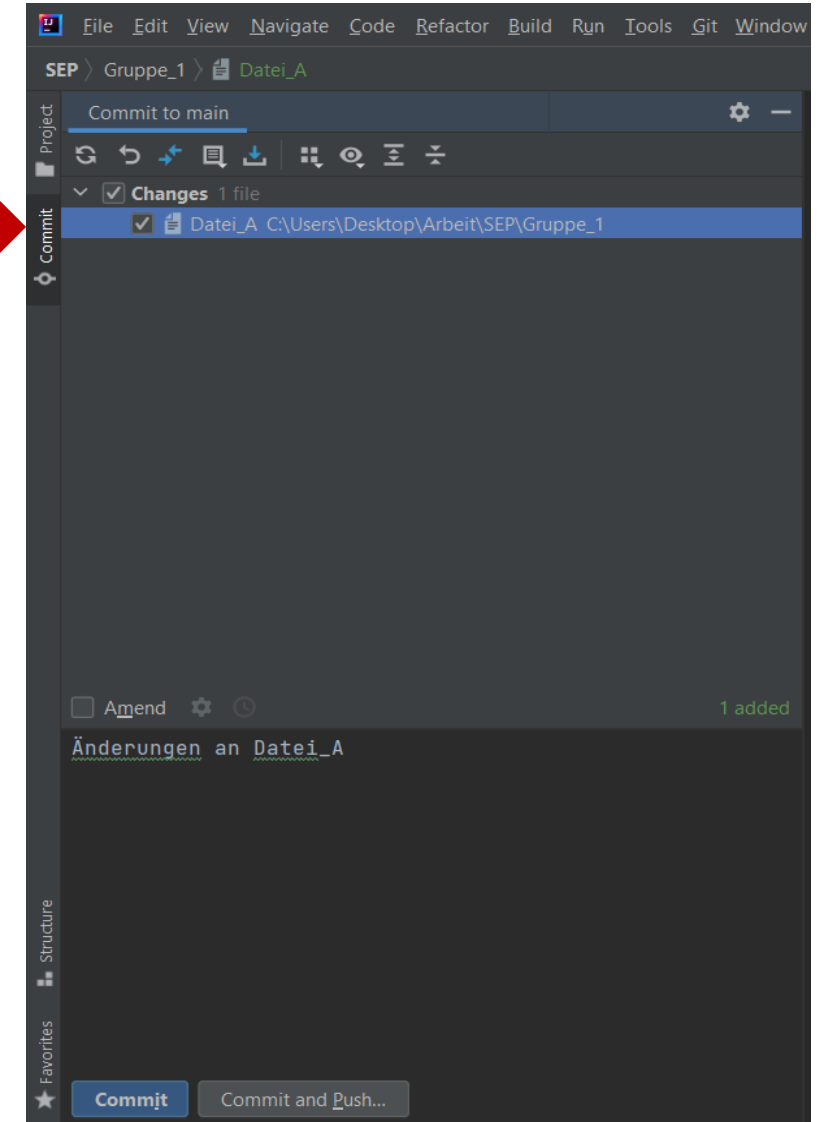


```
MINGW64:/c/Users/Desktop/Arbeit/SEP/Gruppe_1
MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 377 bytes | 377.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://git.uni-due.de/sep/Wintersemester_2021-22/Gruppe_1.git
4640564..0bed591  main -> main
```

Ablauf - Änderungen veröffentlichen

IntelliJ

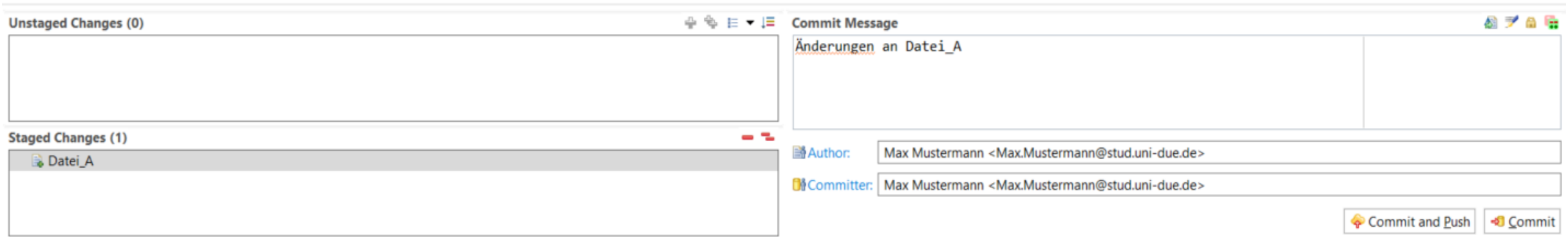
- IntelliJ erkennt automatisch die Dateien, an denen Veränderungen vorgenommen wurden
- Mittels **Setzen** eines **Hakens** werden die gewünschten Änderungen zu einem Commit zusammengefasst
- Im unteren Bereich kann das vorhandene Textfeld genutzt werden, um die vorgenommenen Änderungen zu dokumentieren (Die Nachricht wird **Commit-Message** genannt und ist nach dem Veröffentlichen für andere einsehbar)
- Die Änderungen werden über den *Commit*-Button in das lokale Repository übermittelt



Ablauf - Änderungen veröffentlichen

Eclipse

> Gruppe_1 [main]

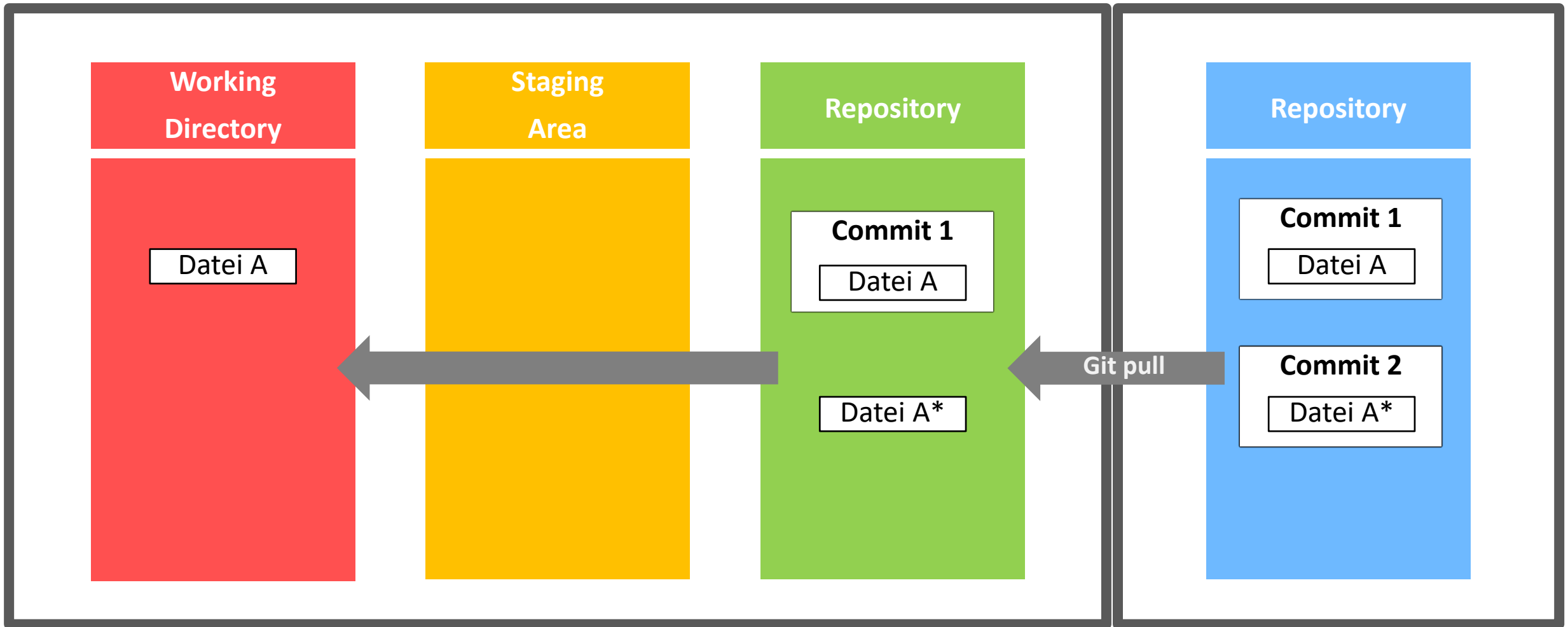


- Eclipse erkennt automatische Änderungen und zeigt an, ob diese sich im **Staging-Bereich** befinden
- Das Textfeld **Commit Message** kann mit sinnvollen Beschreibungen der vorgenommenen Änderungen gefüllt werden (Die Nachricht wird Commit-Message genannt und ist nach dem Veröffentlichen für andere einsehbar)
- Die Änderungen werden über den *Commit*-Button in das lokale Repository übermittelt

Ablauf - pull

LOCAL

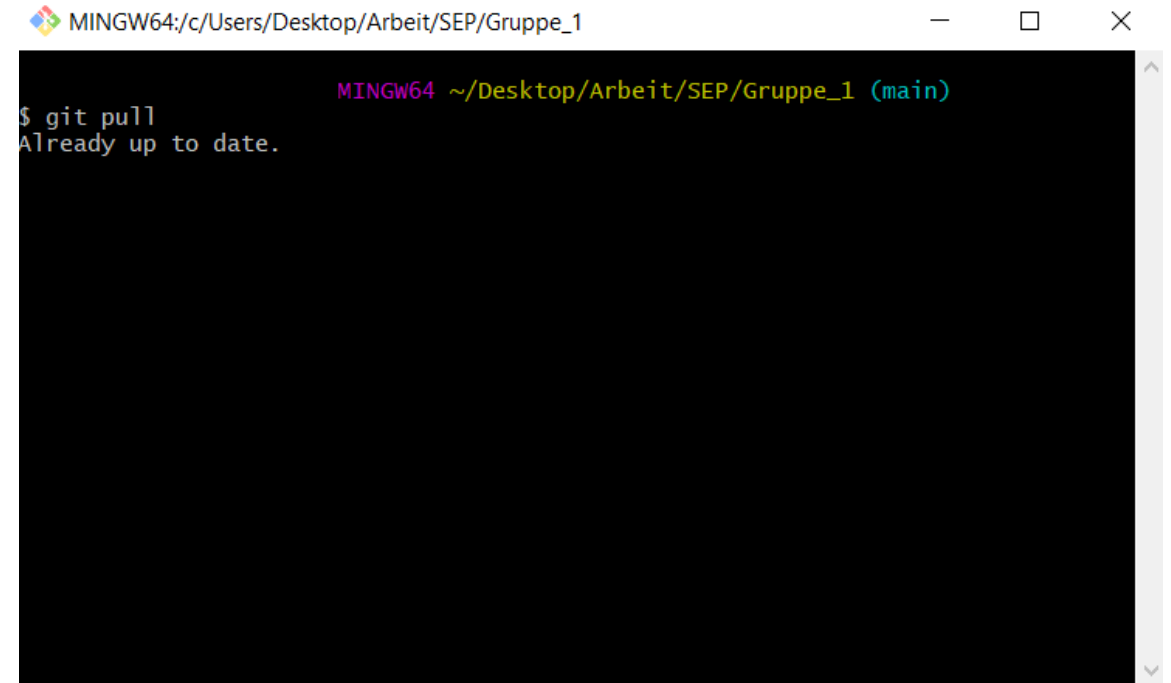
REMOTE



Ablauf - pull

Git Bash

- Mit dem Befehl **git pull** werden die neusten Änderungen aus dem Online Repository heruntergeladen
- Vorher muss gegebenenfalls die **Uni-Kennung** und das **Passwort** eingegeben werden



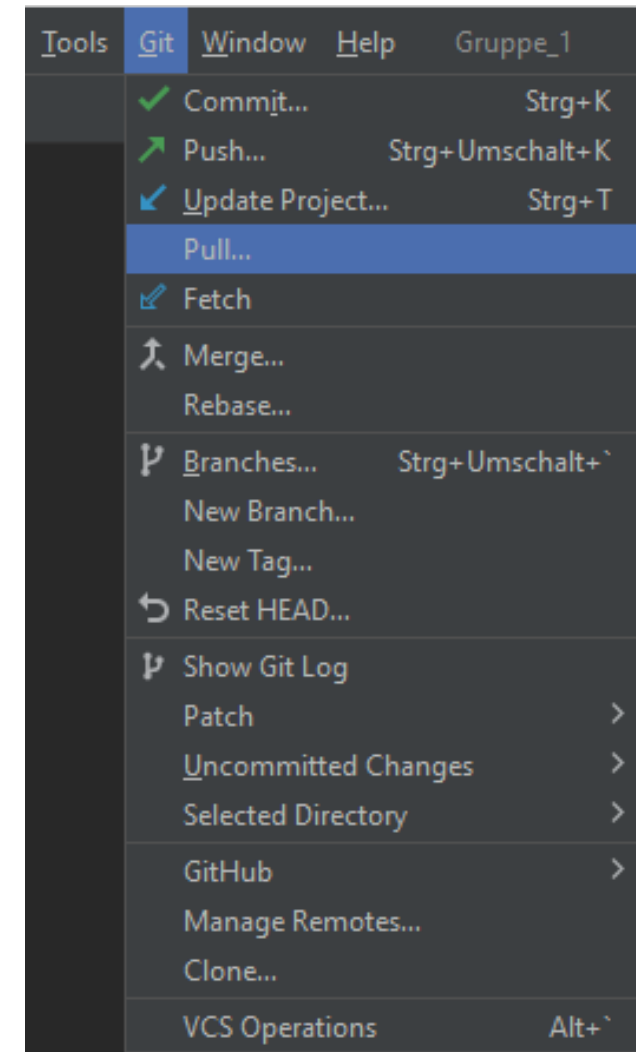
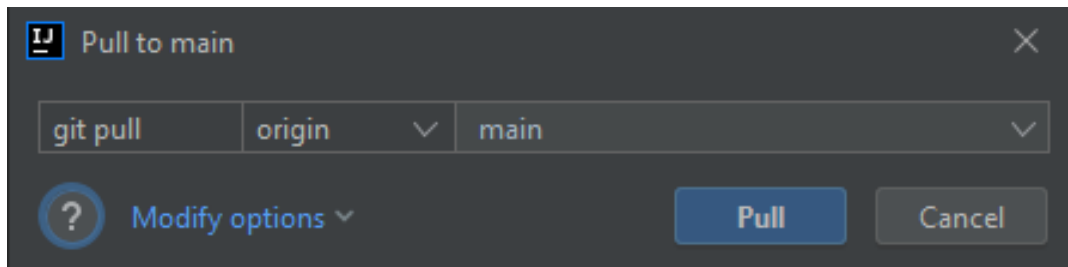
The screenshot shows a terminal window titled 'MINGW64: c:/Users/Desktop/Arbeit/SEP/Gruppe_1'. The prompt is 'MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)'. The user has entered '\$ git pull' and the output is 'Already up to date.'.

```
MINGW64: c:/Users/Desktop/Arbeit/SEP/Gruppe_1
MINGW64 ~/Desktop/Arbeit/SEP/Gruppe_1 (main)
$ git pull
Already up to date.
```

Ablauf - pull

IntelliJ

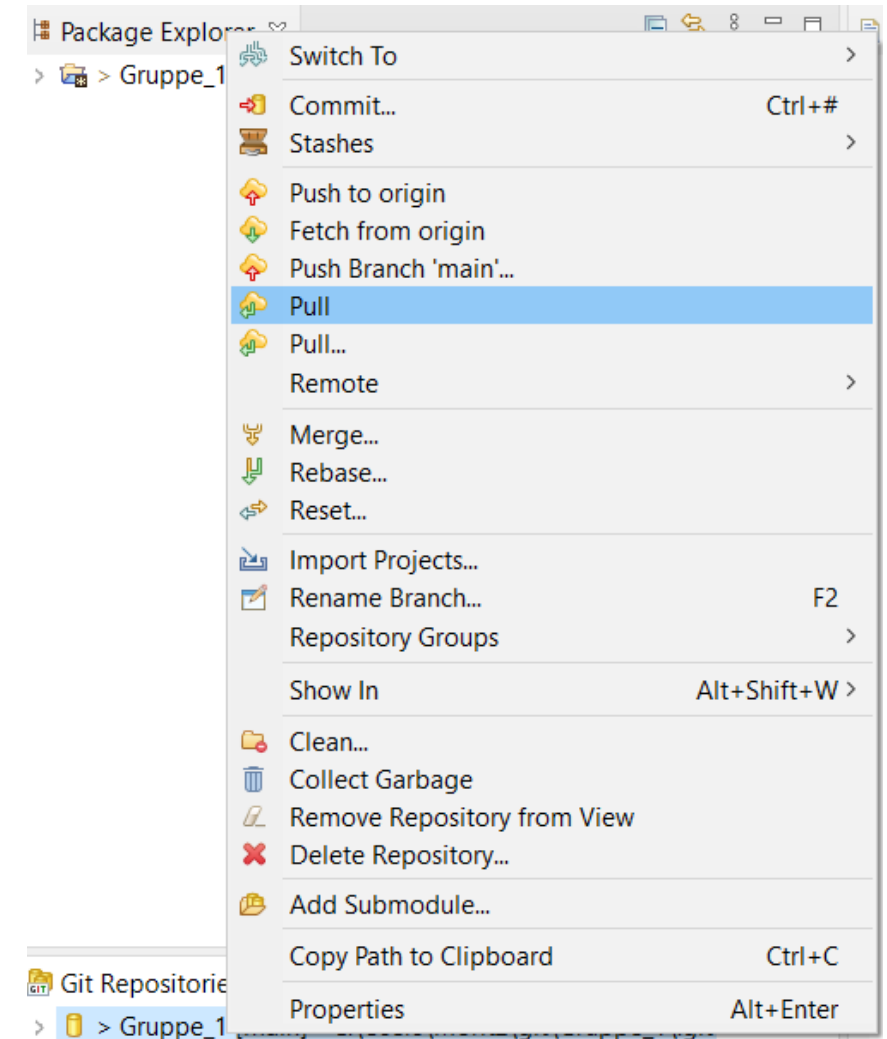
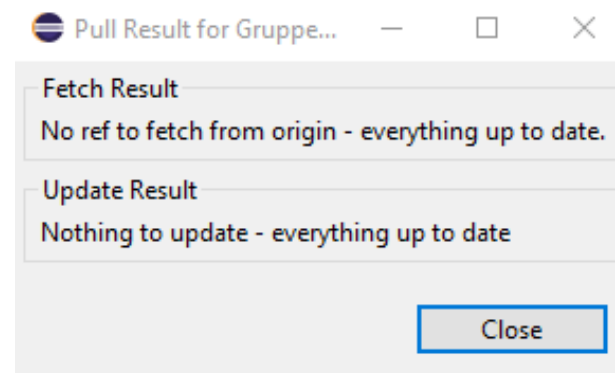
- Über den Reiter *Git* öffnet sich ein Drop-Down Menü mit der Option *Pull...*
- In dem sich öffnenden Fenster den Branch spezifizieren, von dem gepullt werden soll (Default Branch siehe Abb.)
- Nach der Bestätigung über den *Pull*-Button werden die neusten Änderungen heruntergeladen



Ablauf - pull

Eclipse

- Mit einem Rechtsklick auf das entsprechende Repository im Git-Fenster öffnet sich ein Drop-Down Menü
- Die Option *Pull* auswählen und darauf folgend **Uni-Kennung** und **Passwort** eingeben
- Abschließend werden die übernommenen Änderungen angezeigt



Agenda

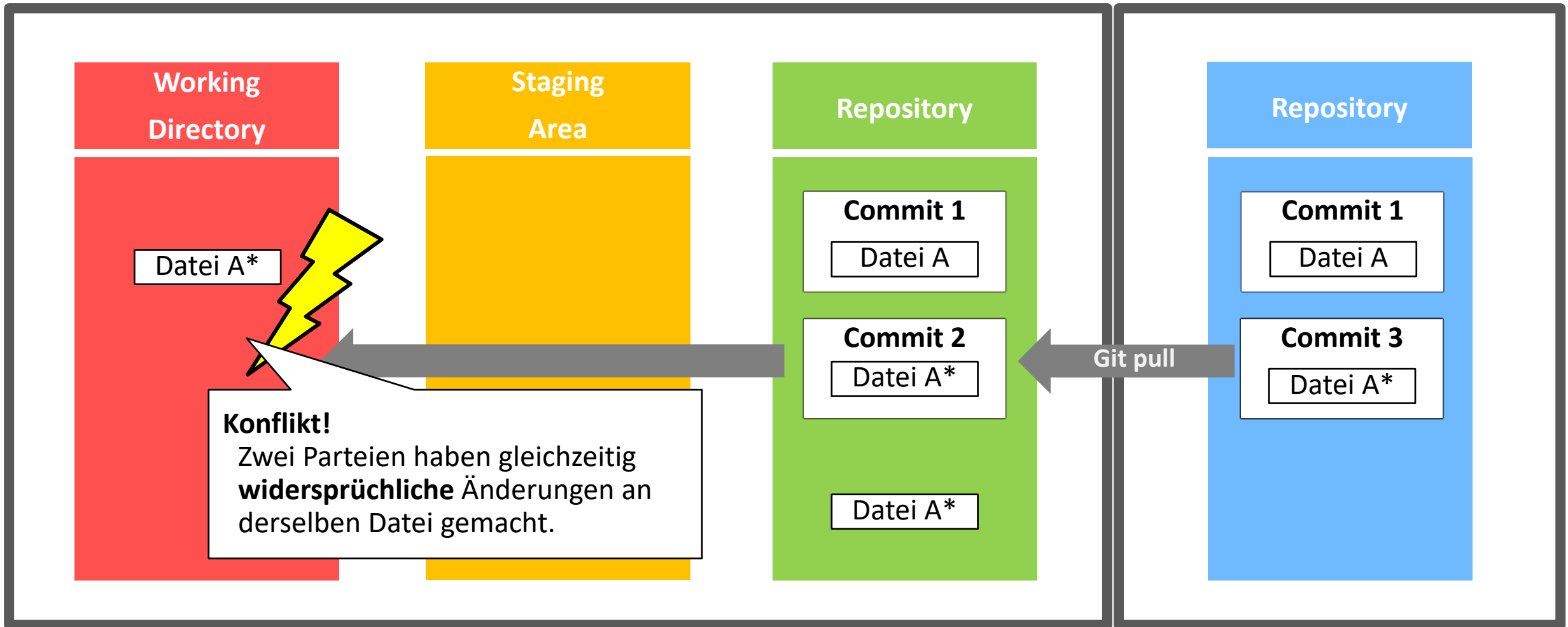
1. Motivation
2. Git – Grundlagen
3. Git – typischer Ablauf
4. **Git – typische Konflikte**
5. GitLab der Universität Duisburg-Essen



Beispiel - Konflikte

LOCAL

REMOTE



- Gleichzeitige Änderungen von Dateien führt zu Konflikten
- Auflösung durch folgende Abfolge:
 1. Konflikt entsteht durch merge/pull Befehl
 2. Konflikt lokal auflösen (Passende Tools auf Folie 39)
 3. Auflösende Änderungen hinzufügen (**add**)
 4. Auflösende Änderungen dauerhaft übernehmen (**commit**)
 5. Auflösende Änderungen hochladen (**push**)

Konflikte – Wie vermeiden?

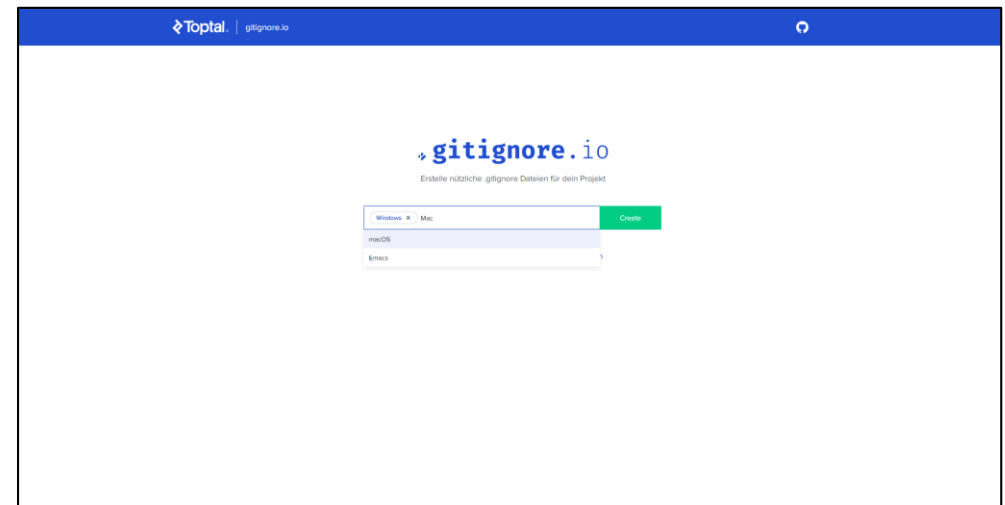
- Konflikte sind **unvermeidlich**
- **Aber:** Ihre Häufigkeit ist reduzierbar!
- Hierzu sollte Folgendes beachtet werden:
 - Arbeiten in eigenen Branches
 - Den Master-Branch sauber halten
 - Mergen der Branches nur bei Fertigstellung der aktuellen Arbeit
 - Master-Branch sollte stets eine lauffähige Version der Software beinhalten

- Erstellt eine exakte Kopie des Master-Branch
 - Der Master-Branch ist der Hauptzweig in jedem Git und wird automatisch erzeugt, wenn ein neues Git-Repository erstellt wird
- Ermöglicht es, unabhängig vom Master-Branch zu arbeiten und zu testen
- Abzweigen und Zusammenführen ist beliebig oft möglich

Befehle:

- **branch** Erstellung eines neuen Branch
- **checkout** Wechsel in einen Branch
- **merge** Zusammenführung von Abzweigungen

- In der Datei .gitignore können Dateien angegeben werden, welche nicht mit gepusht werden sollen
- Beispiele sind z.B. betriebssystemabhängige Dateien, IDE-Konfigurationen, Logfiles, CSV-Dateien und allgemein endgerätspezifische Dateien (Datenbank-Backups, etc.)
- Solche Daten können unnötigen Traffic und Speicherplatz verbrauchen, sowie zu Fehlern führen
- Tipp: www.gitignore.io
 - Hier kann eine gitignore Datei generiert werden, die typische Dateien berücksichtigt
- Weitere Informationen:
<http://git-scm.com/docs/gitignore>



Agenda

1. Motivation
2. Git – Grundlagen
3. Git – typischer Ablauf
4. Git – typische Konflikte
5. **GitLab der Universität
Duisburg-Essen**



GitLab der Uni-DUE

- Zu finden unter: <https://git.uni-due.de>
- Bietet eine Übersicht über Repositories und Projektmanagement



Gitlab Server der Universität Duisburg-Essen

UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

Ein Dienst des ZIM (Zentrum für Informations-und Mediendienste)

[Impressum](#)

[Kontakt: git@uni-due.de](mailto:git@uni-due.de)

LDAP	Standard
LDAP Username	
<input type="text"/>	
Password	
<input type="password"/>	
<input type="checkbox"/> Remember me	
<input type="button" value="Sign in"/>	

Folgendes ist **erforderlich** und **verpflichtend(!)**:

- Anmeldung im GitLab der Uni-DUE
- Klonen des Gruppen-Repositories
- Die globalen Einstellungen von Git so anpassen, dass **jeder** Commit mit dem eigenen Namen und der eigenen studentischen E-Mail unterzeichnet wird
 - Kommandozeilenbefehle, mit denen man die globalen Einstellungen setzen kann:
 - `git config --global user.name "vorname nachname"`
 - `git config --global user.email vn.nn@stud.uni-duisburg-essen.de`
 - Auch alle anderen gängigen Git-Clients erlauben das Setzen der globalen Einstellungen

- Auch wenn Änderungen an dem Repository über den Browser direkt durchgeführt werden, müssen die daraus entstanden Commits mit dem eigenen Namen und der eigenen studentischen E-Mail unterzeichnet werden
- Dies kann nach dem Login auf <https://git.uni-due.de/profile> bearbeitet werden
- **Das Nicht-Umsetzen der zuvor beschriebenen Verpflichtungen kann dazu führen, dass die **erbrachten Leistungen nicht anerkannt** werden!**

Folgende Software erleichtert den Umgang mit Git-Repositories und ermöglicht einen Zugriff über eine integrierte GUI

- Management von Spezifikation im Git:
 - GitKraken
 - Kostenlos in der Standardversion für Studenten der Universität verfügbar (Windows, Mac und Linux)
 - Strukturierte, leicht verständliche Benutzeroberfläche
 - Umfangreich dokumentiert
 - Weitere: TortoiseGit, Sourcetree
- Management von Code im Git:
 - IDEs stellen entsprechende Funktion bereit
 - Projekt einmal clonen und anschließend direkt Code via IDE in Git comitten/pushen und pullen
 - Tutorials für Integration in Eclipse und IntelliJ siehe **Moodle!**

Die folgende Webseite bietet sich an, um den Umgang mit Git zu üben und den eigenen Wissensstand weiter zu vertiefen:

- https://learngitbranching.js.org/?locale=de_DEGitKraken

Hinweis: Zum erfolgreichen Bestehen dieses Modules und einer reibungslosen Zusammenarbeit untereinander solltet Ihr in der Lage sein, die Kapitel „Einführung“ und „Push & Pull“ problemlos zu bearbeiten



- Universität Duisburg-Essen (2017). *Git*. Retrieved from <https://git.uni-due.de>
- Scott Chacon (2014). *Pro Git*. Retrieved from <https://git-scm.com/book/en/v2>
- Git – der einfach Einstieg <https://rogerdudler.github.io/git-guide/index.de.html>
- Git – Tutorials <https://www.atlassian.com/git/tutorials/git-bash>

Verwendete Grafiken

- Grafiken von <https://thenounprojekt.com/>
 - Question by unlimicon
 - Delete Fork by Dimitry Baranovskiy

Vielen Dank für Eure Aufmerksamkeit