

Einführung zu Git

Das Nötigste für die Studienarbeit
im Modul Datenkommunikation

Ege Inanc

Warum ist ein Versionskontrollsystem für die Studienarbeit nützlich?

- Arbeitet man im Team, kann es passieren, dass ein Entwickler den durch einen anderen Entwickler veränderten Sourcecode versehentlich überschreibt
- Ein Versionskontrollsystem oder im Speziellen Git hilft, dies zu verhindern
- Konflikte durch nebenläufige Veränderungen im Sourcecode können erkannt werden und auch ein Rücksetzen auf einen vorherigen Stand ist möglich

Was ist Git?

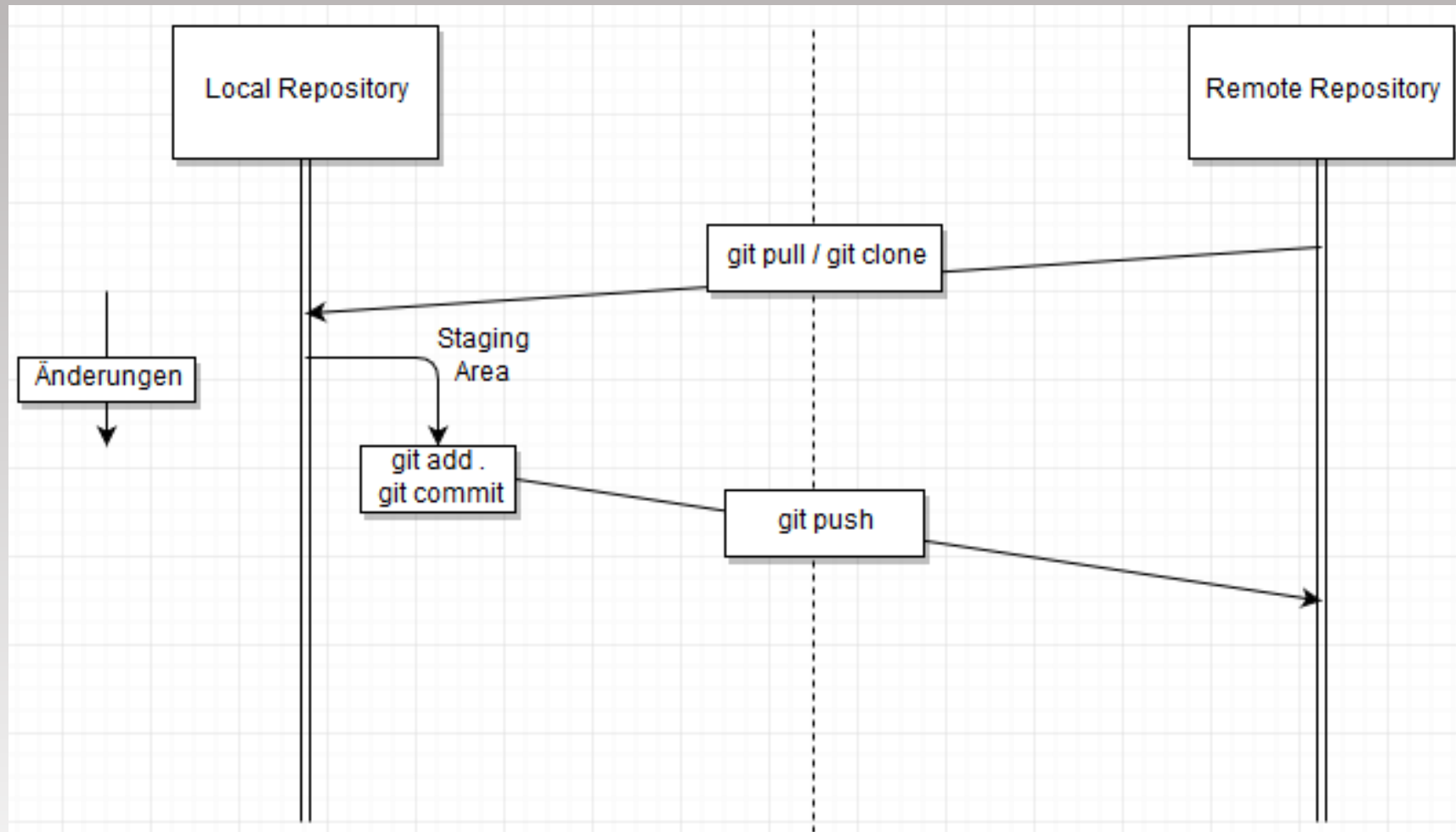
- Git ist ein verteiltes Versionskontrollsystem für Softwareprojekte (Distributes Version Control System, DVCS)
- Jeder Projektmitarbeiter legt sein eigenes lokales Repository an
- Ein lokales Arbeiten auf dem Entwicklungsrechner ist ohne permanente Verbindung zu einem Repository-Server möglich
- Ein Abgleich mit anderen Repositories ist auch möglich, Konflikte werden erkannt

Anm.: Ein Repository ist eine Art Datenbank für den Sourcecode eines Projekts

Verschiedene Repositories in Git

- Local Repository
 - Versionsverwaltung innerhalb des lokalen Verzeichnisses (mittels „git init“)
- Staging Area
 - Temporäre, lokale Lagerungen der festgeschriebenen Änderungen eines Entwicklers
- Remote Repository
 - Versionsverwaltung über eine zentrale Schnittstelle, in der alle Arbeiten der Entwickler zusammengeführt werden

Zusammenhänge



Warum Git ?

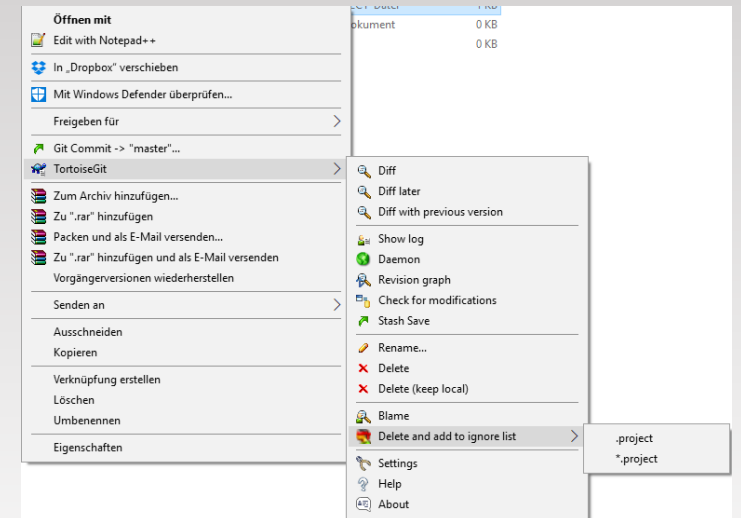
- Relativ schnelles Verständnis der einzelnen Vorgänge ist möglich
- Das Arbeiten ist offline möglich
 - Im Gegensatz zu andere Versionskontrollsystemen, wie z. B. SVN, benötigt man bei Git **keine** ständige Verbindung zum Remote Repository
- Gute Dokumentation

.gitignore (1)

- Dateien eines Projektes, die lokale Bedeutung haben und daher nicht versioniert werden sollen, können in einer .gitignore-Datei gespeichert werden. Sie werden dann von den Git-Befehlen ignoriert
- Bei TortoiseGit (später genauer) ist dies sehr einfach gestaltet
- Hierzu wird eine .gitignore-Datei im lokalen Repo erstellt

Gitignore für Eclipse und IntelliJ:

<https://www.gitignore.io/api/eclipse%2Cintellij>

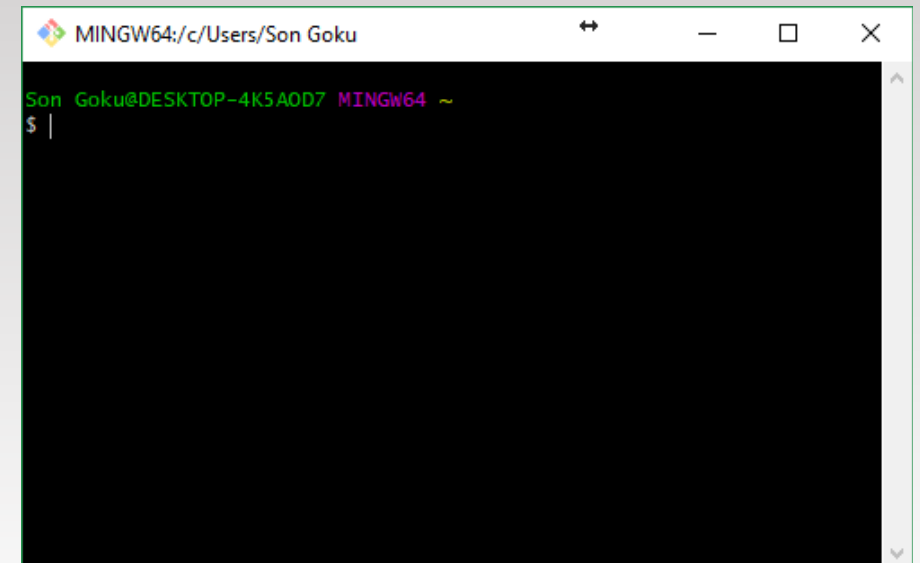


.gitignore (2)

- Unter www.gitignore.io könnt ihr eine gitignore-Datei für die IDEs erstellen lassen
- Ihr könnt bestimmte Dateinamen eintragen oder aber auch wildcards
 - Beispiel: *.log -> es werden alle .log Dateien ignoriert
- Mit einem Eintrag <verzeichnisname>/* werden alle Dateien im Verzeichnis ignoriert
 - verzeichnisname gibt den Namen des Verzeichnisse an
- Mit dem Zeichen „#“ fügt ihr einen Kommentar hinzu

Git Bash

- Erforderliches Tool ist die Git Bash
 - Download unter: <https://git-scm.com/downloads>
 - Zur Zeit in der Version 2.14.2



Wichtigste Befehle

- git init / git clone
- git pull (= git fetch + git merge)
- git push
- git commit
- git status
- git add
- git branch

Repository anlegen

Grundsätzlich zwei Möglichkeiten:

1. `git init`: Ein neues lokales Repository wird angelegt
2. `git clone`: Ein bestehendes remote Repository wird in das lokale Verzeichnissystem geklont

git init

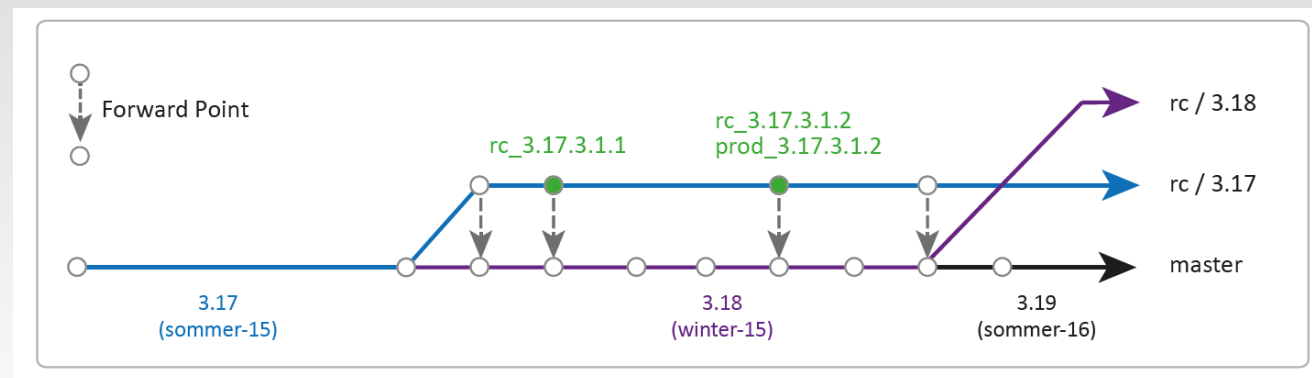
- Im Systemverzeichnis wird die Bash gestartet
- Der Befehl „git init“ initialisiert ein Grundgerüst für ein lokales Repo
- Auf www.github.com ein entferntes Repo anlegen
- Anschließend folgenden Befehl aufrufen:
„git remote add origin <https://github.com/username/reponame.git>“
mit
 - username = eigener Benutzername
 - reponame = Name des entfernten Repositories
- Ersten push durchführen mit „git push –u origin master“

git clone

- Mit Hilfe des Befehls „git clone“ erstellt dann jedes weitere Teammitglied eine Arbeitskopie und ein lokales Repo aus dem entfernte Repo auf www.github.com
- „git clone <https://github.com/pfad-zum-repo/reponame.git>“
mit
 - reponame = Name des entfernten Repositories
- Z. B. „git clone <https://github.com/projekt/dako.git>“

Was ist ein Branch (1)?

- Mit einem Branch (Zweig) kann die Entwicklung fortgesetzt werden, ohne die aktuelle Version zu verändern
- Jedes Projekt hat initial einen Master-Branch
- Während der Entwicklung werden weitere Branches angelegt
- Veränderungen des Sourcecodes in Branches können wieder zurückgeführt werden. Das nennt man „Merging“
- Beispiel:



Was ist ein Branch (2)?

- In Branches entwickelt man also Projekterweiterung ohne den aktuellen Master-Branch zu verändern
- Beispielsweise:
 - SimpleChat wird im Master-Branch verwaltet
 - Für den AdvancedChat wird ein neuer Branch angelegt
- Somit wird der SimpleChat nie berührt, die Teilaufgabe 2 wird im AdvancedChat-Branch entwickelt.
- Achtung: Man muss immer prüfen, auf welchen Branch man gerade arbeitet! -> Dies sieht man in den Klammern

```
Son Goku@Der-Gerät MINGW64 ~/Desktop/Testrepo (master)
$ git checkout -b AdvancedChat
Switched to a new branch 'AdvancedChat'

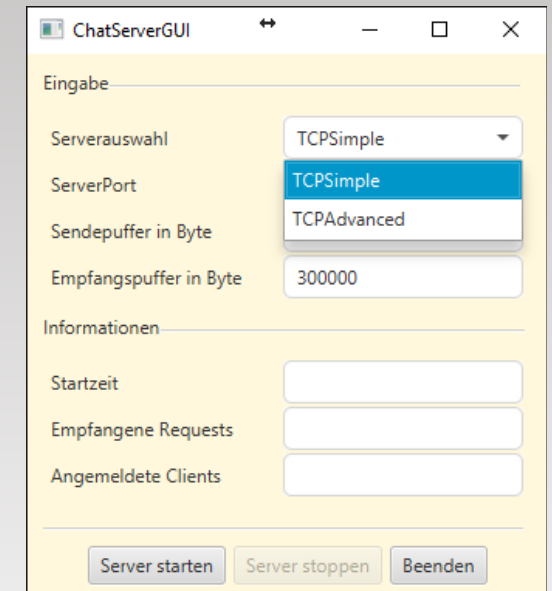
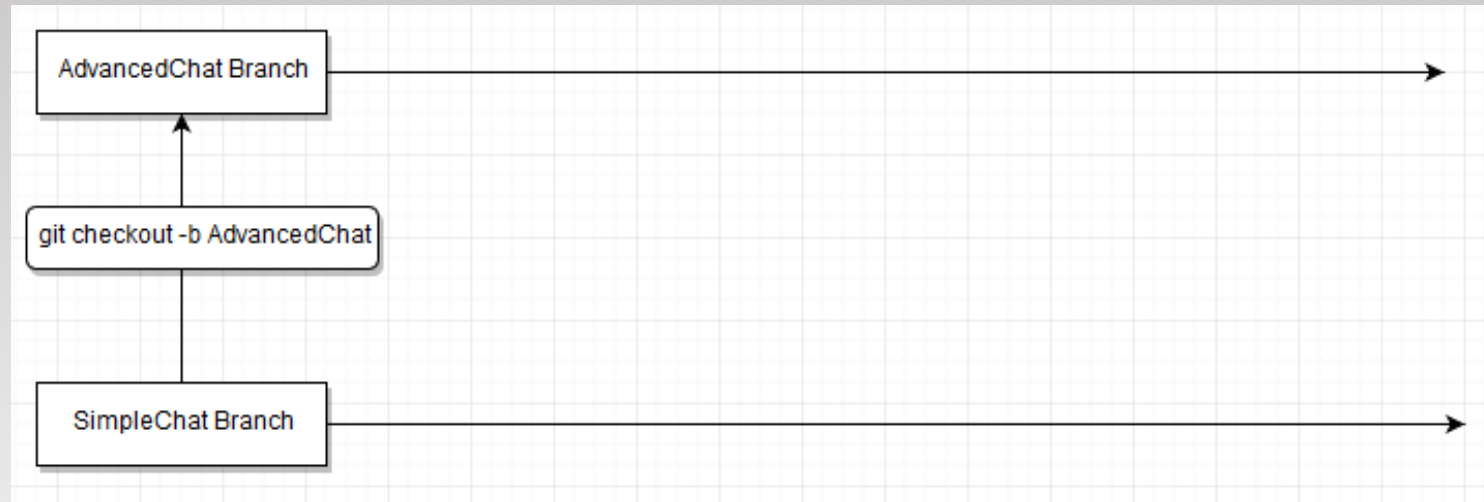
Son Goku@Der-Gerät MINGW64 ~/Desktop/Testrepo (AdvancedChat)
$ |
```

Branching am Chat-Beispiel (1)

- Auf Basis des SimpleChats wird ein neuer Branch erstellt
 - „git checkout –b AdvancedChat“
- Erstellen eines weiteren Branches auf Basis des AdvancedChat, wenn die Entwicklung aufgeteilt wird (z. B. Client- und Serverentwicklung), z. B.:
 - „git checkout –b AdvancedChatServer“
 - „git checkout -b AdvancedChatClient“
- Hier arbeitet ein Entwickler am Server auf einem eigenen Branch und ein weiterer auf einem eigenen Branch für den Client
- Die Basis des SimpleChat ist somit zu jeder Zeit lauffähig
- Wechseln innerhalb der Branches dann mit
 - „git checkout <branchname>“

Branching am Chat-Beispiel (2)

- Am Ende des Projekt ist es sinnvoll zur Übung dann die Branches miteinander zu mergen (später mehr)



- Unter anderem auch um die Combobox der GUI zu erweitern

git pull (= git fetch + git merge)

- Bevor man anfängt, weiter zu arbeiten, sollte der aktuelle Stand aus dem Repository besorgt werden, um Konflikte möglichst zu vermeiden
- Mit „git pull“ werden die Änderungen vom entfernten Repo in den lokalen branch geladen und gemerged
- Merging ist das Zusammenführen von neuen Codeteilen in den lokalen Stand

git status

- Dieser Befehl zeigt alle Änderungen im lokalen Repo im Gegensatz zum Remote Repo an

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    NeueDatei.txt

nothing added to commit but untracked files present (use "git add" to track)
```

- Mit git add <dateiname> (oder git add . für alle Files) werden diese für den Commit vorbereitet

```
Son Goku@DESKTOP-4K5A0D7 MINGW64 ~/Desktop/Test (master)
$ git add .

Son Goku@DESKTOP-4K5A0D7 MINGW64 ~/Desktop/Test (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   NeueDatei.txt
```

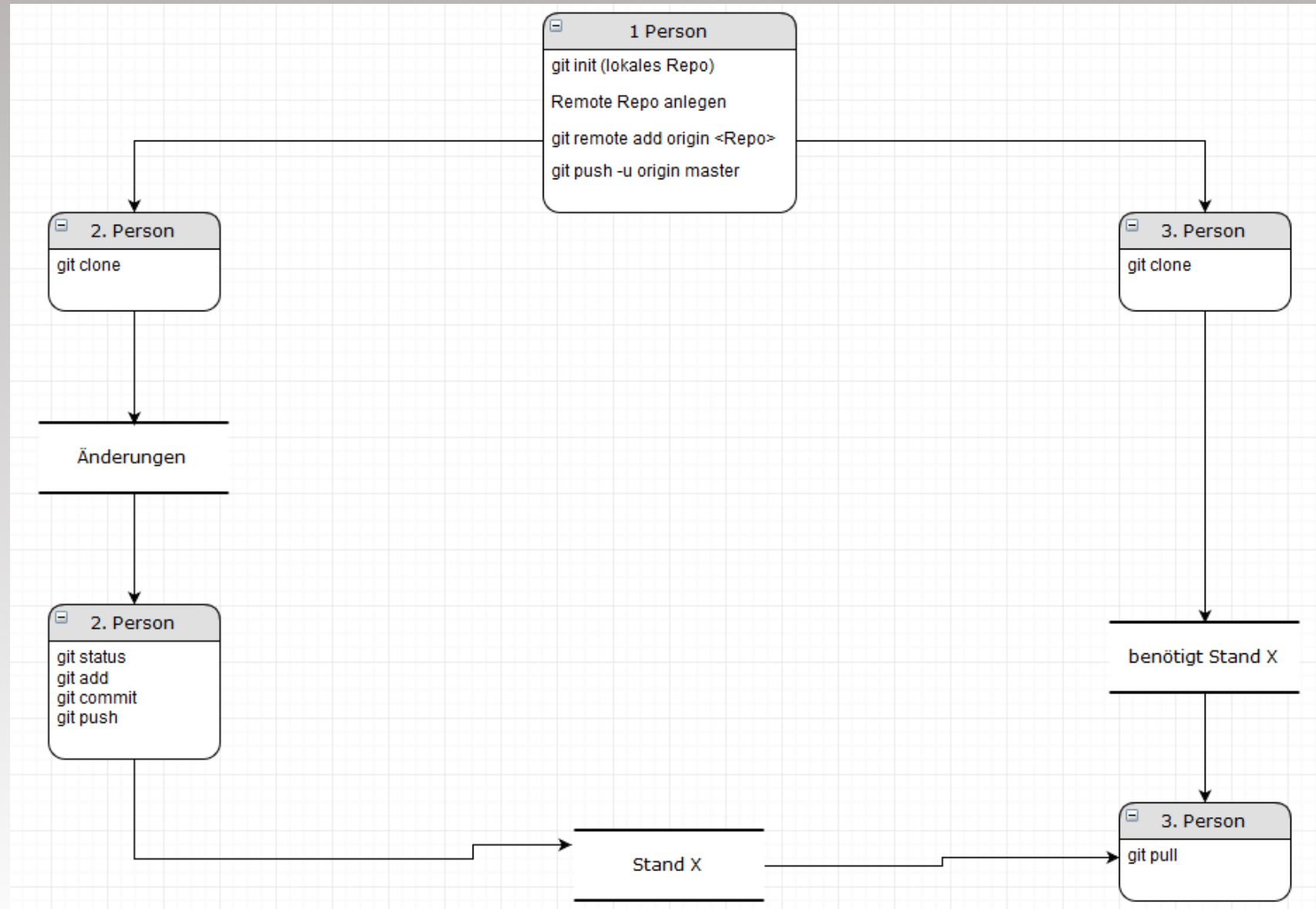
git commit

- Mit dem Kommando `git commit -m „<commit message>“` schreibt Ihr Eure Änderungen fest
 - commit message = beliebiger, sinnvoller Kommentar, um den Entwicklungsstand zu erläutern.
- Dann ist alles für den „push“ in das entfernte Repo vorbereiten

git push

- Nach dem Commit wird der Befehl „git push“ ausgeführt, um die Änderungen für die anderen Entwickler verfügbar zu machen
- Diese können sich dann diese Änderungen mit „git pull“ auf ihr lokales Repo kopieren

Allgemeiner Ablauf



git branch

- Ein Branch ist ein weiterer Sourccode-Zweig im Projekt
- Wird ein Repo erzeugt wird der Master-Branch erstellt
- Mit „git branch <branchname>“ wird ein neuer lokaler Branch erstellt
- „git checkout <branchname>“ wechselt in den neuen Branch
- „git merge <branchname>“ verschmelzt die Inhalte der Branches
- „git branch“ listet alle lokalen Branches auf
- „git branch -a“ listet alle lokalen und remote Branches auf
- „git checkout -b <branchname>“ erstellt einen neuen Branch und wechselt sofort in diesen

Branches mergen

- Beispiel:
 - Ihr befindet euch im Branch „develop“ habt hier Änderungen getätigt.
 - Ihr wollt diese Änderungen in den Master-Branch übernehmen:
 - 1. git checkout master // Wechsel in den Master-Branch
 - 2. git merge develop // Überführe Änderungen von „develop“ in den Master-Branch

Konflikte

- Konflikte entstehen, wenn in der gleichen Datei die selbe Zeile geändert wird
- In der Regel erscheint beim Push-Versuch dann folgende Nachricht in der Bash

```
$ git push
git push https://github.com/egeinanc/Versuch.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/egeinanc/Versuch.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- Mit „git pull“ wird dann automatisch gemerged, schlägt dies fehl, muss von Hand eine Konfliktbereinigung durchgeführt werden

```
Auto-merging konflikt.txt
CONFLICT (content): Merge conflict in konflikt.txt
Automatic merge failed; fix conflicts and then commit the result.
```

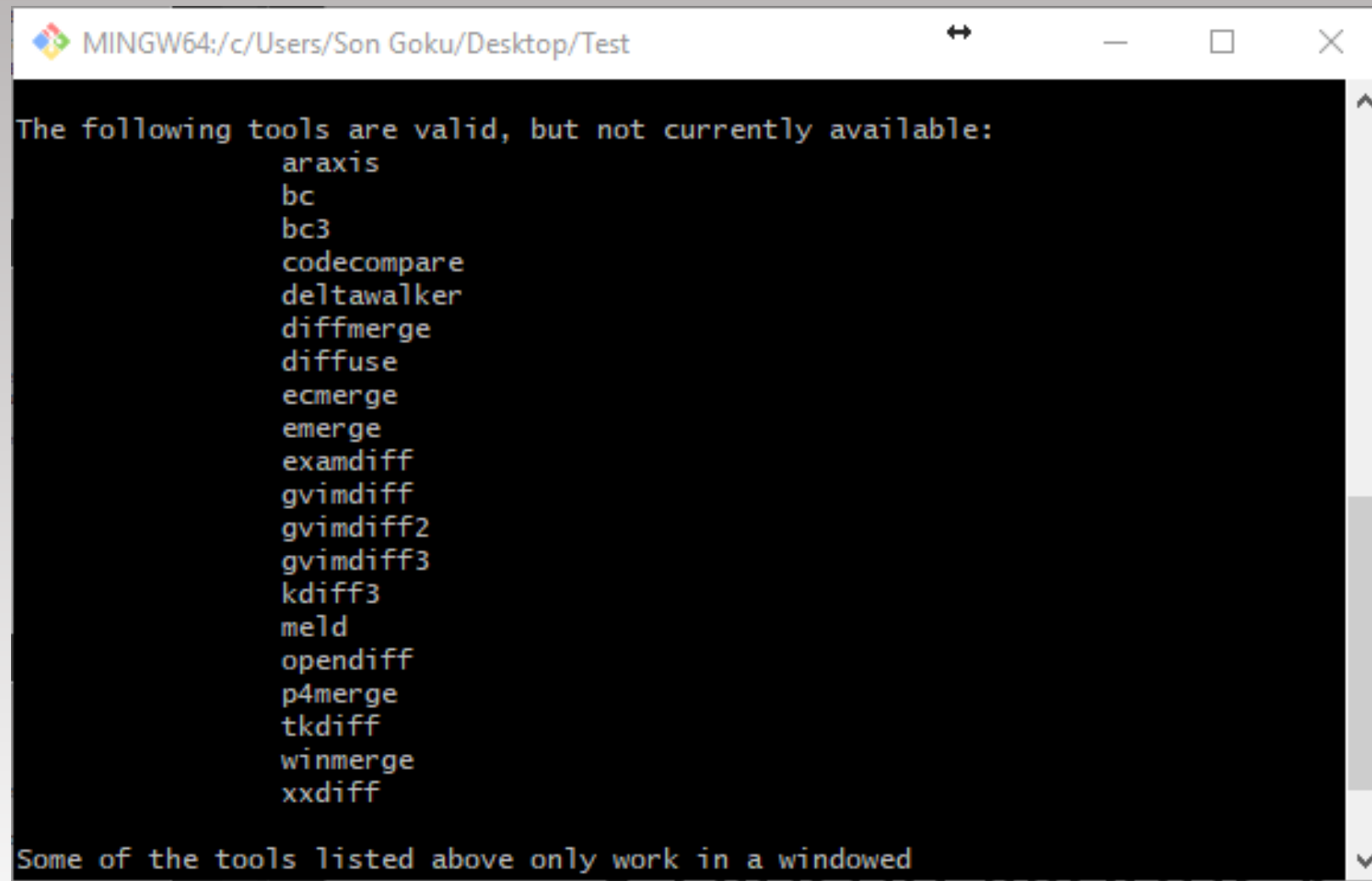
Merging

- Nachdem der Befehl „git pull“ ausgeführt wurde wird der Merge-Modus gestartet. Als Branchname wird dann „MERGING | master“ ausgegeben
- Diesen Modus kann man mit „git merge -- abort“ verlassen
- Es gibt viele verschiedene Merge Tools diese kann man mit dem Befehl
 - „git mergetool -- tool-help“ ausgeben lassen

```
$ git mergetool --tool-help
'git mergetool --tool=<tool>' may be set to one of the following:
    tortoisemerge
    vimdiff
    vimdiff2
    vimdiff3

The following tools are valid, but not currently available:
    araxis
    bc
    bc3
```

Merging Tools



The screenshot shows a terminal window titled "MINGW64:/c/Users/Son Goku/Desktop/Test". The terminal output lists various merging tools that are valid but not currently available. The list includes: araxis, bc, bc3, codecompare, deltawalker, diffmerge, diffuse, ecmerge, emerge, exandiff, gvimdiff, gvimdiff2, gvimdiff3, kdiff3, meld, opendiff, p4merge, tkdiff, winmerge, and xxdiff. A note at the bottom states: "Some of the tools listed above only work in a windowed".

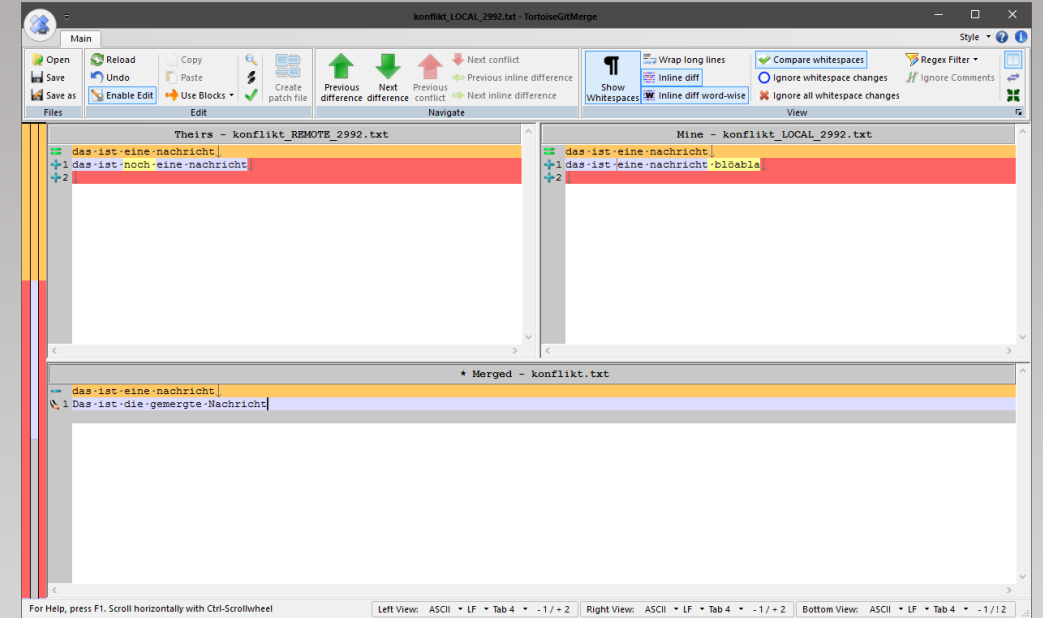
```
MINGW64:/c/Users/Son Goku/Desktop/Test

The following tools are valid, but not currently available:
    araxis
    bc
    bc3
    codecompare
    deltawalker
    diffmerge
    diffuse
    ecmerge
    emerge
    exandiff
    gvimdiff
    gvimdiff2
    gvimdiff3
    kdiff3
    meld
    opendiff
    p4merge
    tkdiff
    winmerge
    xxdiff

Some of the tools listed above only work in a windowed
```

Tortoisemerge

- „git mergetool -- tool=tortoisemerge“



- Bei diesem Tool werden die entfernte und die lokale Version angezeigt
- Unten findet Ihr die „gemergte“ Version

Ergebnis (bei Tortoise)

- Die gemergte Datei wird mit dem gleichen Namen neu erstellt
- Die in Konfliktstehende Datei wird mit <Dateiname>.orig im Repo gespeichert
- Mit „git status“ kann man sich dann wieder die Dateien ansehen
- Schließlich wird dann die gemergte Datei committed und in das Repo gepusht

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
  modified:   konflikt.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  konflikt.txt.orig
```

Nützliche Links

- <http://ohshitgit.com/>
- <https://git-scm.com/>
- <https://www.gitignore.io/>
- <http://rogerdudler.github.io/git-guide/>
- <https://www.git-tower.com/blog/git-cheat-sheet/>

Vielen Dank für eure Aufmerksamkeit

Noch Fragen ?

Online:

- <https://entwickler.de/online/development/git-subversion-svn-versionskontrollsystem-579792227.html>
- <https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts>
- https://de.slideshare.net/RandalSchwartz/introduction-to-git-11451326/5-Why_git_Essential_to_Linux

Offline:

- Daily Git: Wie ein kompetenter Kollege Ihnen Git erklären würde, Martin Dilger (verfügbar in der Bibliothek der HM unter 00/ST 230 D576)