Programmieren Vorkurs

Tag 4 - Schleifen und Git

Nils-André Forjahn, 11.10.2018

Über mich

Nils-André Forjahn

- Studiere Informatik
- Java-HiWi am Institut für Softwaretechnik und Fahrzeuginformatik
- Lehr-HiWi am Institut für Anwendungssicherheit
- Mail: `n.forjahn@tu-braunschweig.de`

Gliederung

- **Schleifen**
  - Motivation
  - for
  - while
  - break
  - continue
- **Git**
  - Warum Git?
  - Struktur
  - Arbeiten mit Git

Schleifen

Motivation

- Menge an Input meist nicht im voraus bekannt
  Beispiel: Überprüfung, ob alle Zahlen einer Menge gerade sind.
- Terminierung des Programms nach Output oft nicht erwünscht
  Beispiel: Internetbrowser, Computerspiele, Betriebssysteme, ...

$\Rightarrow$ Problem: Konstrukt zur wiederholten, steuerbaren Ausführung von Code benötigt!

Schleifen

```
public static void main(String[] args) {
        bool numbersEven = true;

        if(Integer.parseInt(args[0]) % 2 == 1)
                numbersEven = false;

        ...

        System.out.println(
                "All numbers even: " + numbersEven);
}
```

Schleifen

for

- Führt den eingeschlossenen Codeblock so lange aus, wie eine Condition erfüllt ist
- Stellt einen Zähler zur Verfügung, der nach jedem Schleifendurchlauf verändert werden kann, z.B. um 1 hochgezählt
- Ideal zum Arbeiten mit Datenmengen wie Arrays
- Syntax:

```
for(deklaration;condition;manipulation) {
        ...
}
```

Schleifen

Beispiele

```
for(int i = 1; i < 10; i++) ...
```

```
for(int i = 1; i != 16; i = i * 2) ...
```

```
for(int i, j = 1; i < 10; i = i + j + 1) ...
```

Schleifen

```
public static void main(String[] args) {
  bool numbersEven = true;

  for(int i = 0; i < args.Length; i++) {
    if(Integer.parseInt(args[i]) % 2 == 1)
      numbersEven = false;
  }

  System.out.println(
  "All numbers even: " + numbersEven);
}
```

Schleifen

foreach

- Spezielle Abart von `for`
- Verzichtet auf einen Zähler und stellt direkt nach jedem Durchlauf das nächste Datenobjekt zur Verfügung
- Reihenfolge der Datenobjekte ist nicht garantiert
- Syntax:

```
for(deklaration: datenmenge) {
...
}
```

Schleifen

Beispiele

```
for(String argument: args) ...

for(int zahl: zahlenArray) ...
```

Schleifen

```
public static void main(String[] args) {
  bool numbersEven = true;

  for(String argument: args) {
    if(Integer.parseInt(argument) % 2 == 1)
      numbersEven = false;
  }

  System.out.println(
    "All numbers even: " + numbersEven);
}
```

Schleifen

while

- Führt den eingeschlossenen Codeblock so lange aus, wie eine Condition erfüllt ist
- Stellt keinerlei Zähler oder andere Variablen zur Verfügung
- Eignet sich besonders zur Realisierung von nichtterminierenden Programmen
- Syntax:

```
while(condition) {
        ...
}
```

Schleifen

Beispiele

```
while(!eingabe.Equals("exit")) ...
```

```
while(i < 10) ...
```

Schleifen

```
public static void main(String[] args) {
  Scanner scanner = new Scanner();
  String input = scanner.nextLine();
  bool numbersEven = true;

  while(!input.Equals("stop")) {
    if(Integer.parseInt(input) % 2 == 1)
        numbersEven = false;
    input = scanner.nextLine();
  }

  System.out.println(
    "All numbers even: " + numbersEven);
```

Schleifen

do.. while

- Unterart von `while`
- Führt wie `while` den eingeschlossenen Codeblock so lange aus, wie eine Condition erfüllt ist
- Jedoch **mindestens einmal!**

```
do {
...
} while(condition)
```

Schleifen

Beispiele

```
do {...} while(!eingabe.Equals("exit"))
```

```
do {...} while(i < 10)
```

Schleifen

break

- Ermöglicht das vorzeitige Verlassen einer Schleife
- Wirkt bei verschachtelten Schleifen auf die zuletzt benutze
- Beispiel

```
while(true) {
    input = scanner.nextLine();
    if(input.Equals("exit"))
        break;

    System.out.println("Last input: " + input);
}
```

Schleifen

continue

- Überspringt den Rest des momentanen Schleifendurchlaufs, jedoch **nicht** die gesamte Schleife!
- Wirkt wie `break` bei verschachtelten Schleifen auf die zuletzt benutze
- Beispiel

```
for(i = 0; i < 20; i++) {
   if(i % 2 == 1)
      continue;

   System.out.println(i);
}
```

Schleifen

```
public static void main(String[] args) {
  Scanner scanner = new Scanner();
  do {
    String input = scanner.nextLine();
    if(input.Equals("stop"))
      break;
    if(Integer.parseInt(input) % 2 == 1) {
      numbersEven = false;
  } while(true)

  System.out.println(
    "All numbers even: " + numbersEven);
}
```

Schleifen

Zusammenfassung

- Schleifen ermöglichen das wiederholte Ausführen von Code
- Die Schlüsselwörter `break` und `continue` ermöglichen das Abbrechen oder Überspringen von Schleifen
- Es gibt verschiedene Arten von Schleifen, die sich für bestimmte Aufgaben besser eignen:
  `for` für z.B. Datenmengen, `while` für kontinuierlichen Input

→ Es gibt oft mehrere Möglichkeiten eine Schleife zu implementieren

Git

Verteiltes Arbeiten im Team gehört zum Alltag in der Softwareentwicklung

- Mitglieder können sich auf verschiedenen Kontinenten befinden, u.U. ohne Internetzugang
- Verschiedene Personen könnten auf den selben Dateien arbeiten
- Falsche Änderungen wie das löschen oder verändern von kritischem Code könnte das gesamte Projekt sabotieren

$\rightarrow$ Gelöst durch Git!

Git

Warum Git?

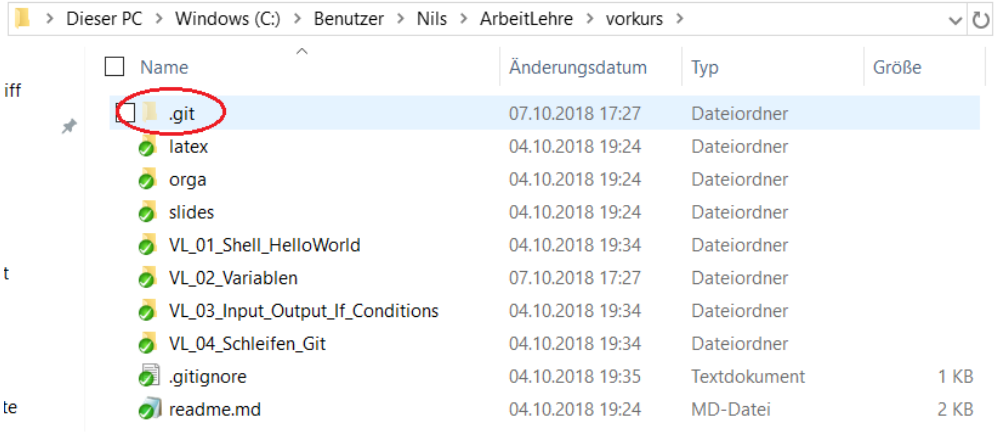
- Freies, verteiltes Versionssystem
- Seit 2005 in der Entwicklung
- Besonders im Open-Source-Bereich verbreitet (siehe Github)
- Erlaubt das Branchen von Projekten
- Wird außerdem zur **Abgabe der Programmierenhausaufgaben** genutzt!

Git

Struktur

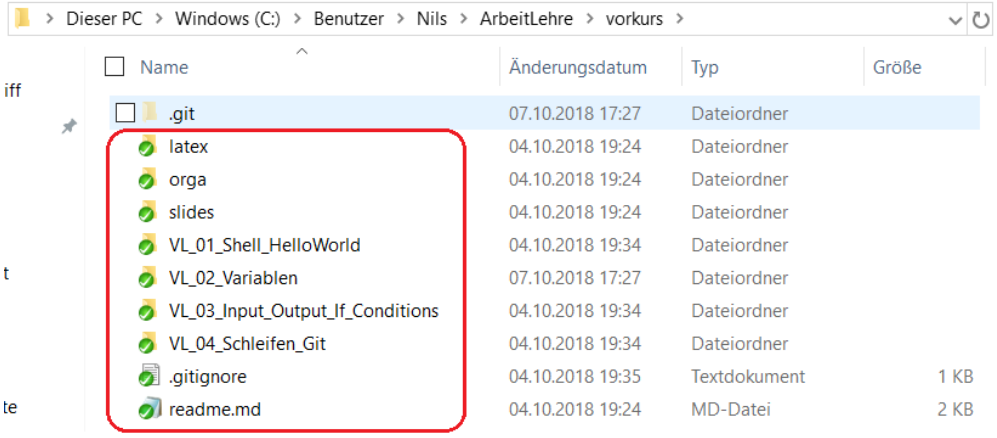
- Der Verlauf eines Projekts wird im `Repository` gespeichert
- Jeder Nutzer hat seine eigene Version des Projekts (`Working Copy`) und Repositories (`Local Repository`)
- Meist gibt es ein zentrales Repository (`Remote Repository`) für das Projekt, das jeder Nutzer mit seinem abgleicht
- Änderungen am Projekt werden in der `Staging Area` erfasst und in `Commits` festgehalten

Git



Local Repository

Git



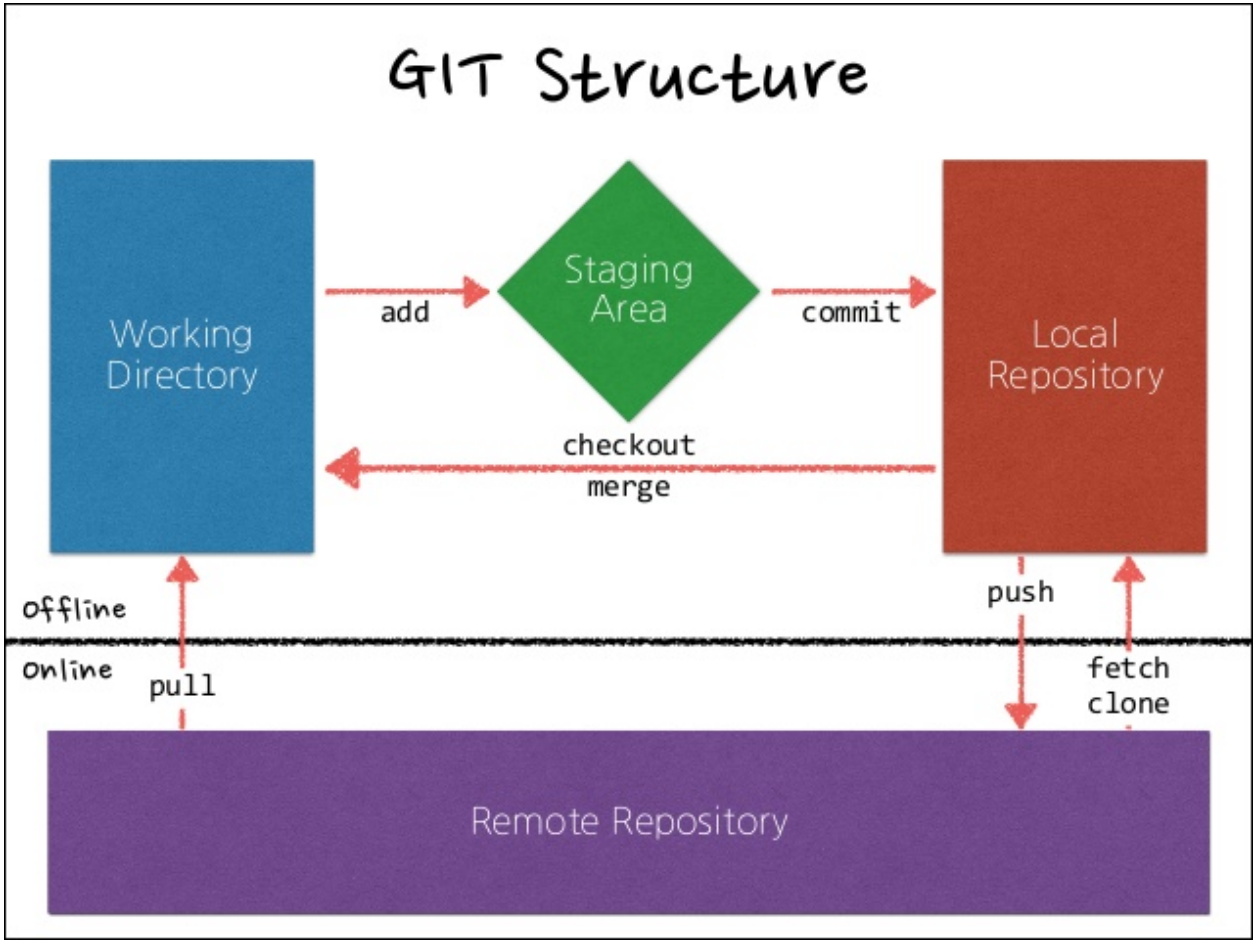
Working Copy

Git

Arbeiten mit Git

- Klonen eines Remote Repositories mittels
  `git clone <PfadZumRemoteRepo>`
- Erfassen von Dateiänderungen mit
  `git add <Dateiname>`
- Festhalten von Änderungen mit
  `git commit -m «Kommentar>"`
- Übertragen von Commits zum Remote Repository durch
  `git push origin master`
- Beziehen und Umsetzen von Commits vom Remote Repository durch
  `git pull`

Git

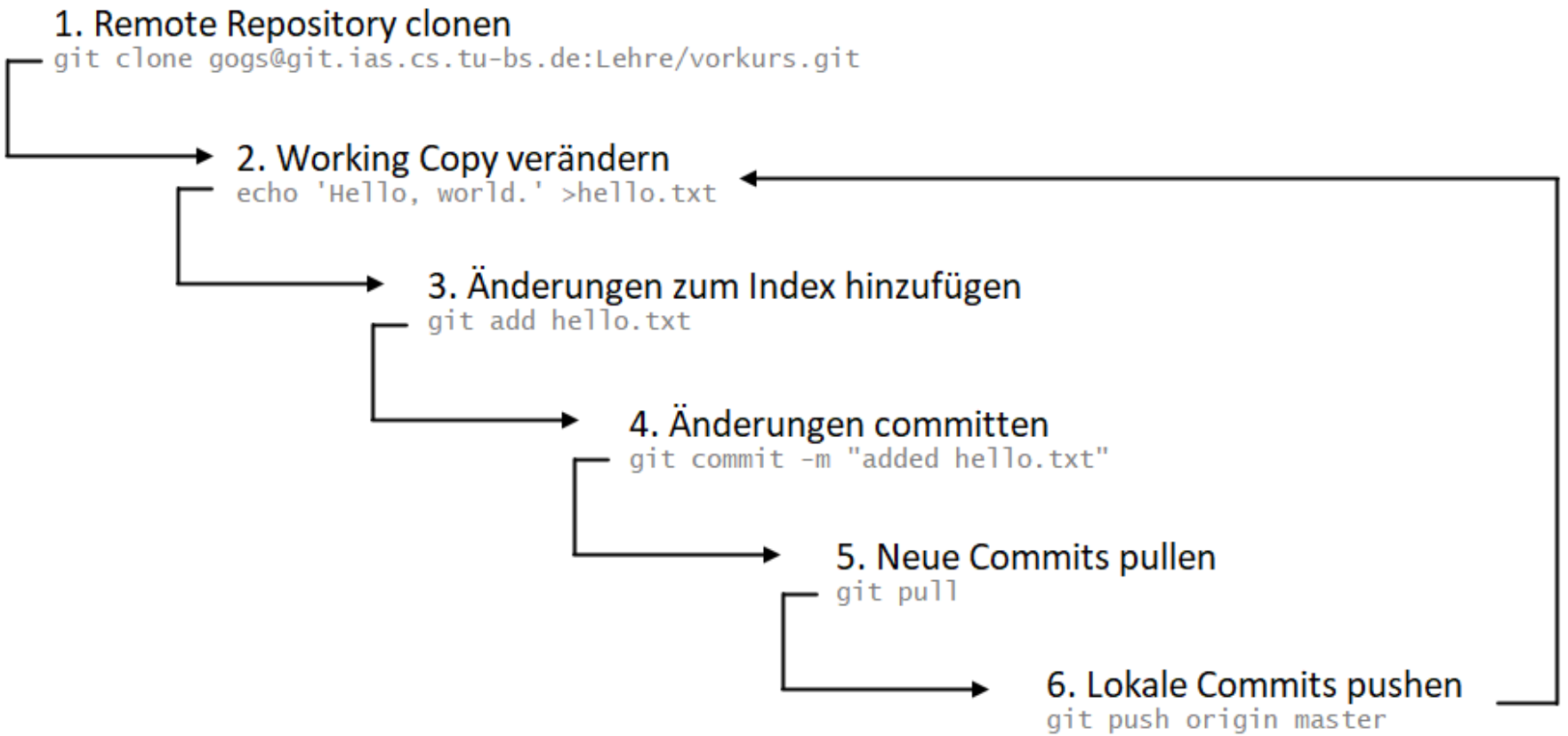

GIT Structure
Working Directory
add
Staging Area
commit
Local Repository
checkout
merge
offline
online
pull
push
fetch
clone
Remote Repository

Git

Weitere Befehle

- Erzeugen eines neuen Respositories
  `git init`
- Updaten des Local Repository ohne Working Copy
  `git fetch origin`
- Einarbeiten der Änderungen im Local Repository in die Working Copy
  `git merge`
- Und wenn man mal nicht weiter weiß…
  `git help`

Git


1. Remote Repository clonen
git clone gogs@git.ias.cs.tu-bs.de:Lehre/vorkurs.git
2. Working Copy verändern
echo 'Hello, world.' >hello.txt
3. Änderungen zum Index hinzufügen
git add hello.txt
4. Änderungen committen
git commit -m "added hello.txt"
5. Neue Commits pullen
git pull
6. Lokale Commits pushen
git push origin master

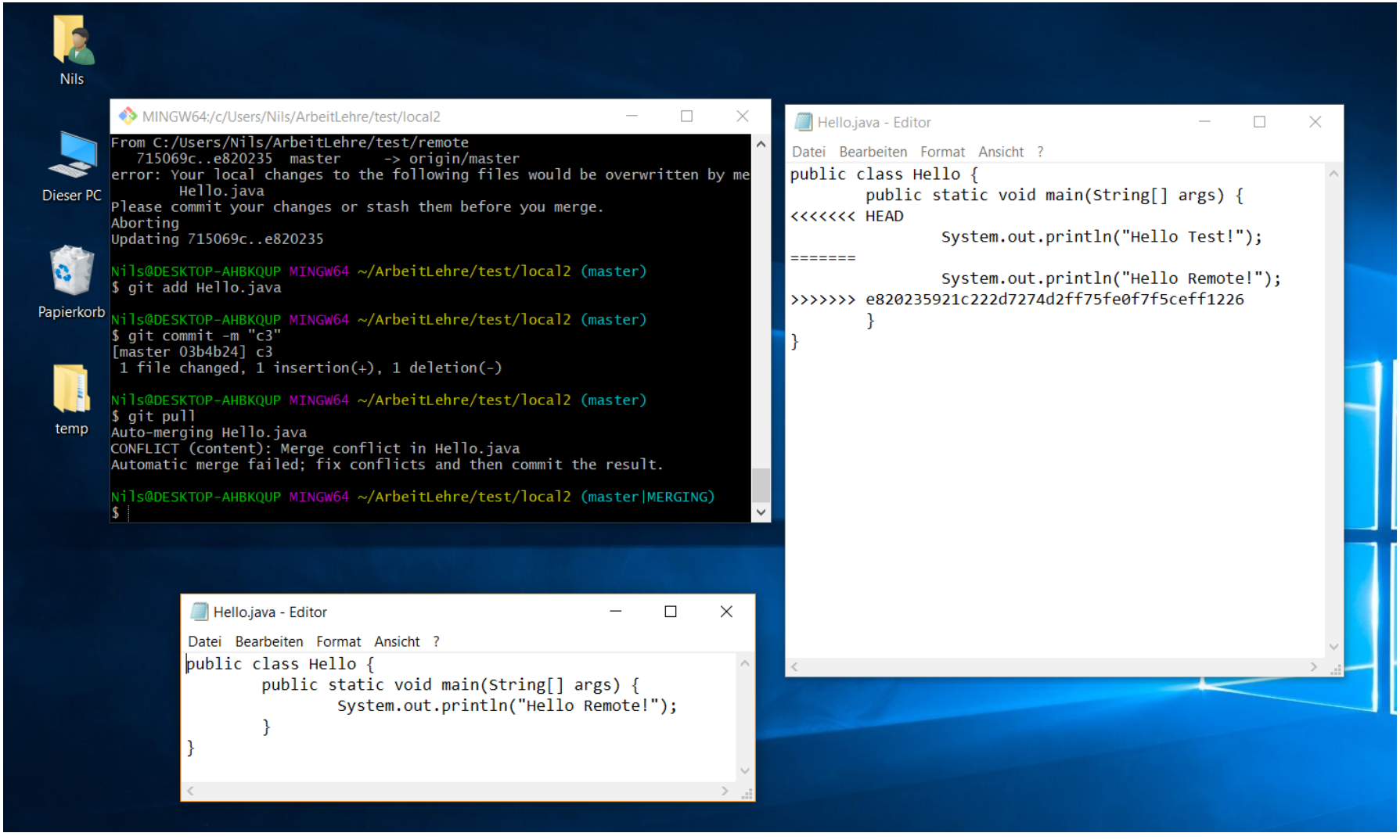
Git

Konflikte

- Es kann passieren, dass ein Commit Dateien ändert, die man lokal verändert hat
- In vielen Fällen kann Git diesen Konflikt auflösen
- Bei Änderung z.B. der selben Codezeile ist keine automatische Auflösung möglich, manche Konflikte müssen per Hand gelöst werden

$\rightarrow$ Lösung durch Modifikation und erneutem Hinzufügen der Konfliktdatei!

Git


Nils
Dieser PC
Papierkorb
temp
MINGW64:/c/Users/Nils/ArbeitLehre/test/local2
From C:/Users/Nils/ArbeitLehre/test/remote
715069c..e820235 master -> origin/master
error: Your local changes to the following files would be overwritten by me
Hello.java
Please commit your changes or stash them before you merge.
Aborting
Updating 715069c..e820235
Nils@DESKTOP-AHBKQUP MINGW64 ~/ArbeitLehre/test/local2 (master)
$ git add Hello.java
Nils@DESKTOP-AHBKQUP MINGW64 ~/ArbeitLehre/test/local2 (master)
$ git commit -m "c3"
[master 03b4b24] c3
1 file changed, 1 insertion(+), 1 deletion(-)
Nils@DESKTOP-AHBKQUP MINGW64 ~/ArbeitLehre/test/local2 (master)
$ git pull
Auto-merging Hello.java
CONFLICT (content): Merge conflict in Hello.java
Automatic merge failed; fix conflicts and then commit the result.
Nils@DESKTOP-AHBKQUP MINGW64 ~/ArbeitLehre/test/local2 (master|MERGING)
$
Hello.java - Editor
Datei Bearbeiten Format Ansicht ?
public class Hello {
public static void main(String[] args) {
<<<<<<< HEAD
System.out.println("Hello Test!");
=======
System.out.println("Hello Remote!");
>>>>>>> e820235921c222d7274d2ff75fe0f7f5ceff1226
}
}
Hello.java - Editor
Datei Bearbeiten Format Ansicht ?
public class Hello {
public static void main(String[] args) {
System.out.println("Hello Remote!");
}
}

Abschluss

Vielen Dank für eure Aufmerksamkeit!
Ich hoffe ihr seid noch wach. :)