ETH zürich

Eprog - Session 01, Git und EBNF

Jonas Wetzel

25.09.2024

Plan für heute

Organisatorisches
Wieso eigentlich EBNF
Beispiele EBNF
Follow-up
Einrichten von Git
EBNF Kahoot
Outro

n.ethz.ch/~jwetzel

Website

E-Mail

1. Organisatorisches

Organisatorisches

- Mein Name: Jonas Wetzel
- Bei Fragen: jwetzel@ethz.ch

Organisatorisches

- Mein Name: Jonas Wetzel
- Bei Fragen: jwetzel@ethz.ch
- Neue Aufgaben: Mittwoch Morgen (im Normalfall)

Organisatorisches

- Mein Name: Jonas Wetzel
- Bei Fragen: jwetzel@ethz.ch
- Neue Aufgaben: Mittwoch Morgen (im Normalfall)
- Abgabe der Übungen bis Dienstag Abend (23:59) Folgewoche

Organisatorisches

- Mein Name: Jonas Wetzel
- Bei Fragen: jwetzel@ethz.ch
- Neue Aufgaben: Mittwoch Morgen (im Normalfall)
- Abgabe der Übungen bis Dienstag Abend (23:59) Folgewoche
  - Abgabe immer via Git
  - Lösungen in separatem Projekt auf Git

Organisatorisches

- Wir haben eine WhatsApp Gruppe

Organisatorisches

- Wir haben eine WhatsApp Gruppe
- alles was ich behandle könnt ihr auf meiner Website finden

2. Wieso eigentlich EBNF

Wofür wir EBNF brauchen

Wofür wir EBNF brauchen

- Computer verstehen nur Maschinencode, welcher schwer ist zu lesen für uns Menschen

Wofür wir EBNF brauchen

- Computer verstehen nur Maschinencode, welcher schwer ist zu lesen für uns Menschen
- Um Programmierung einfacher zu machen, haben wir deswegen Sprachen wie Java entwickelt

Wofür wir EBNF brauchen

- Computer verstehen nur Maschinencode, welcher schwer ist zu lesen für uns Menschen
- Um Programmierung einfacher zu machen, haben wir deswegen Sprachen wie Java entwickelt
- jedoch muss unsere Sprache am Ende in Maschinensprache übersetzt werden

Wofür wir EBNF brauchen

- Das Werkzeug was uns dabei hilft nennt sich Compiler

Wofür wir EBNF brauchen

- Das Werkzeug was uns dabei hilft nennt sich Compiler
- Um eine exakte Übersetzung von unserer Sprache, in die des Computers zu ermöglichen, brauchen wir präzise Syntax

Wofür wir EBNF brauchen

- Das Werkzeug was uns dabei hilft nennt sich Compiler
- Um eine exakte Übersetzung von unserer Sprache, in die des Computers zu ermöglichen, brauchen wir präzise Syntax
- dabei hilft uns EBNF

Wofür wir EBNF brauchen

- Mit EBNF können wir überprüfen, ob unser Code grammatikalisch richtig ist

Wofür wir EBNF brauchen

- Mit EBNF können wir überprüfen, ob unser Code grammatikalisch richtig ist

  <Anweisung> ::= <Zuweisung> | <Bedingung>
  <Zuweisung> ::= <Variable> "=" <Wert>
  <Bedingung> ::= "if" <Ausdruck> "then" <Anweisung>

Wofür wir EBNF brauchen

- Backus hat die Entwicklung der ersten weit verbreitete Programmiersprache Fortran geleitet

Wofür wir EBNF brauchen

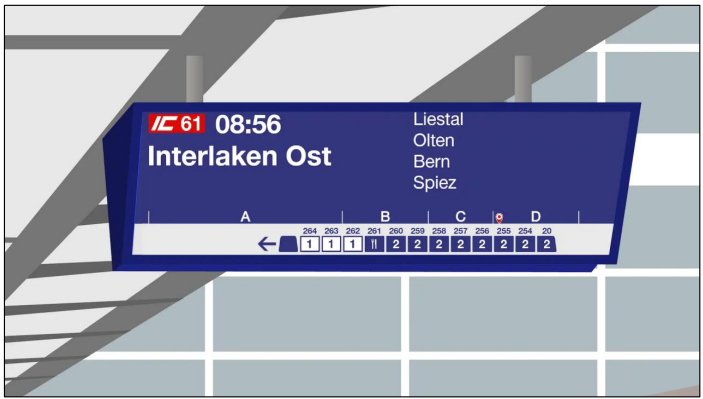
- Backus hat die Entwicklung der ersten weit verbreitete Programmiersprache Fortran geleitet
- Naur hat am ersten Compiler gearbeitet und die Algol Sprache mitentwickelt

Wofür wir EBNF brauchen

- Backus hat die Entwicklung der ersten weit verbreitete Programmiersprache Fortran geleitet
- Naur hat am ersten Compiler gearbeitet und die Algol Sprache mitentwickelt
- beide Turing Award

3. Beispiele EBNF

EBNF – Einfaches Beispiel – Bahnhof



Quelle: SBB

EBNF – Einfaches Beispiel – Bahnhof

EBNF – Einfaches Beispiel – Bahnhof

EBNF-Beschreibung von Anzeigetafel

<zugbezeichnung> <= IC61 | RE | S3 | EC
<abfahrtszeit> <= <stunde> : <minute>
<zielbahnhof> <= Zürich HB | Bern | Basel | Wil | Chur | Interlaken Ost
<stunde> = ?
<minute> = ?
<anzeigetafel> <= <zugbezeichnung> <abfahrtszeit> <zielbahnhof>

EBNF – Beispiel Programmiersprache

```
<programm> <= PROGRAM <bezeichner>
              BEGIN
                { <zuweisung> ; }
                END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

Wie prüfe ich, was legal ist?

1. Informeller Beweis
2. Tabellen
3. Ableitungsbaum
4. Graphische Darstellung

JETZT

Quelle:
Wikipedia

```
<programm> <= PROGRAM <bezeichner>
            BEGIN
              { <zuweisung> ; }
              END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

```
PROGRAM DEMO1
BEGIN
        A0:=3;
        B:=+45;
        H:=-100023;
        C:=A;
        D123:=B34A;
        ESEL:=GIRAFFE;
        TEXTZEILE:=”HALLO";
END
```

```
<programm> <= PROGRAM <bezeichner>
            BEGIN
              { <zuweisung> ; }
              END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

```
PROGRAM DEMO1
BEGIN
        A0:=3;
        B:=+45;
        H:=-100023;
        C:=A;
        D123:=B34A;
        ESEL:=GIRAFFE;
        TEXTZEILE:=”HALLO";
END
```

```
<programm> <= PROGRAM <bezeichner>
            BEGIN
              { <zuweisung> ; }
             END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

```
PROGRAM DEMO2
BEGIN
        myVariable3 := 5
        2GOOD2BETRUE := 6ER
END
```

```
<programm> <= PROGRAM <bezeichner>
           BEGIN
            { <zuweisung> ; }
            END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

```
PROGRAM DEMO2
BEGIN
        myVariable3 := 5
        2GOOD2BETRUE := 6ER
END
```

```
<programm> <= PROGRAM <bezeichner>
            BEGIN
              { <zuweisung> ; }
              END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

```
PROGRAM DEMO3
BEGIN
        GOODNAME := +-5
        MARK := 5.5
END
```

```
<programm> <= PROGRAM <bezeichner>
            BEGIN
              { <zuweisung> ; }
              END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

```
PROGRAM DEMO3
BEGIN
        GOODNAME := +-5
        MARK := 5.5
END
```

Repetition: Ableitungen

- Ableitungstabelle
  - Erste Zeile ist Startregel
  - Letzte Zeile ist Zeichenfolge
  - Übergang zwischen zwei Zeilen entspricht Ableitungsschritt

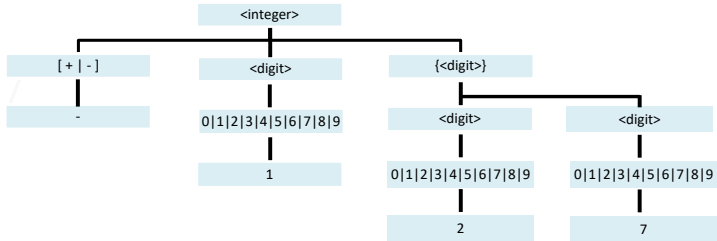
- Ableitungsbaum
  - Wurzel ist Namen der Startregel
  - Blätter sind Zeichen
  - Verbindungen stehen für einen Ableitungsschritt

Beispiel: Ableitung von -127 als Baum

<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer>	⟵	[+ | -] <digit> {<digit>}

Beispiel: Ableitung von -127 als Baum

<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer>	⟵	[+ | -] <digit> {<digit>}



Beispiel: Ableitung von -127 als Tabelle

(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<integer>	⟵	[+ | -] <digit> {<digit>}

Beispiel: Ableitung von -127 als Tabelle

(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<integer>	⟵	[+ | -] <digit> {<digit>}

<integer>	⟵	[+ | -] <digit> {<digit>}	(R2)
	⟵	+ | - <digit> {<digit>}	Option gewählt
	⟵	- <digit> {<digit>}	- gewählt
	⟵	- <digit> <digit> <digit>	2 mal wiederholt
	⟵	- 1 <digit> <digit>	(R1) und 1 gewählt
	⟵	- 1 2 <digit>	(R1) und 2 gewählt
	⟵	- 1 2 7	(R1) und 7 gewählt

EBNF Notation

- In alten Prüfungen wird oft kursiv verwendet für EBNF-Regeln.
- *digit* statt <digit>
- Ab diesem Semester ist eine EBNF-Regel nur korrekt, wenn sie durch < > gekennzeichnet ist.
- Ebenfalls verwenden wir <- statt <=, beides wird aber als korrekt bewertet.

4. Follow-up

Follow-up letzte Woche: Palindrom

- Erstellen Sie eine Beschreibung `<palindrom>`, welche als legale Symbole alle Zahlen zulässt, die von vorne und hinten gleich gelesen werden und die nur die Ziffern von 1 bis 4 verwenden. Beispiele sind 11, 232, 444.

Follow-up letzte Woche: Palindrom

- Erstellen Sie eine Beschreibung `<palindrom>`, welche als legale Symbole alle Zahlen zulässt, die von vorne und hinten gleich gelesen werden und die nur die Ziffern von 1 bis 4 verwenden. Beispiele sind 11, 232, 444.
- **Solution:**

```
<palindrom> ::= [ 1<palindrom>1 | 2<palindrom>2
|3<palindrom>3 | 4<palindrom>4 | 1 | 2 | 3 | 4 ]
```

Follow-up letzte Woche: Five

- Erstellen Sie eine Beschreibung `<five>`, welche alle Summen von positiven Zahlen zulässt, welche 5 ergeben. Beispiele sind “1 + 4”, “2 + 1 + 1 + 1”, “5”.

Follow-up letzte Woche: Five

- Erstellen Sie eine Beschreibung `<five>`, welche alle Summen von positiven Zahlen zulässt, welche 5 ergeben. Beispiele sind “1 + 4”, “2 + 1 + 1 + 1”, “5”.
- **Solution:**

```
<five> ::= 0 + <five> | 1 + <four> | 2 + <three> | 3 + <two>
| 4 + <one> | 5 + <zero> | 5
<four> ::= 0 + <four> | 1 + <three> | 2 + <two> | 3 + <one>
| 4 + <zero> | 4
<three> ::= 0 + <three> | 1 + <two> | 2 + <one> | 3 + <zero>
| 3
<two> ::= 0 + <two> | 1 + <one> | 2 + <zero> | 2
<one> ::= 0 + <one> | 1 + <zero> | 1
<zero> ::= 0 + <zero> | 0
```

Follow-up letzte Woche: OddEight

- Erstellen Sie eine Beschreibung `<oddEight>`, die alle Zahlen enthält, in denen die Ziffer 8 ungerade oft vorkommt. Beispiele sind 8, 128, 8881.

Follow-up letzte Woche: OddEight

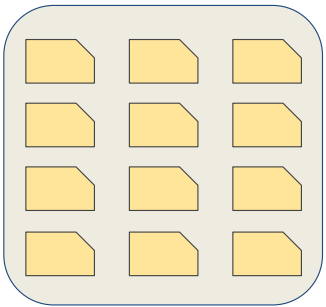
- Erstellen Sie eine Beschreibung `<oddEight>`, die alle Zahlen enthält, in denen die Ziffer 8 ungerade oft vorkommt. Beispiele sind 8, 128, 8881.
- **Solution:**

```
<oddEight> ::= [+|-] <oddEight2>
<oddEight2> ::= [<evenEight>] 8 [<evenEight>]
<evenEight> ::= <noEight> | <oddEight2><oddEight2>
<noEight> ::= (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9)
[<noEight>]
```

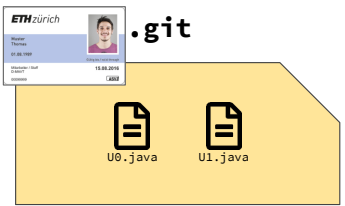
5. Einrichten von Git

Was ist Git?

Git Repository



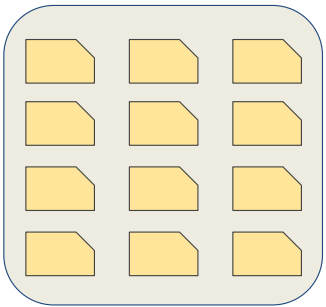
ETH Git-Server



Jedes Repository auf dem Git-Server ist privat

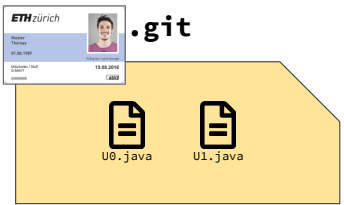
Was ist Git?

Git Repository

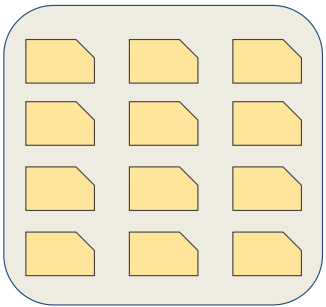


ETH Git-Server

Jedes Repository auf dem Git-Server
Enthält eine Folge von ***Commits*** (die ***History***)



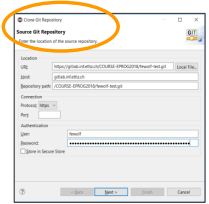
Git Clone: Einmaliges Einrichten

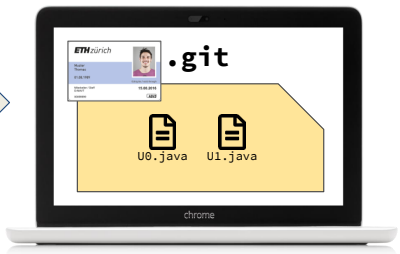


ETH Git-Server

Clone

Kopiert das ganze Repository auf den eigenen Computer

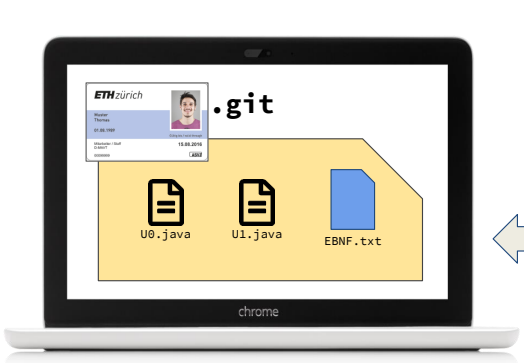


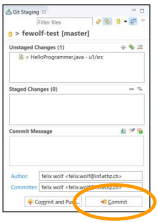


Lokales Git-Repository

Was ist Git?

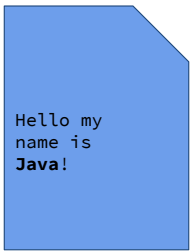
Git Commit: Fortschritt speichern





Commit

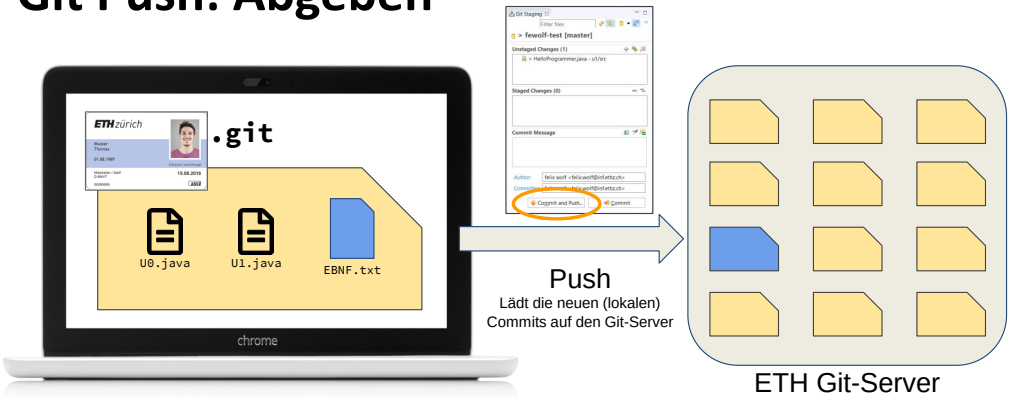
Fügt neuen Commit mit Änderungen/neuen Dateien der *lokalen* History hinzu



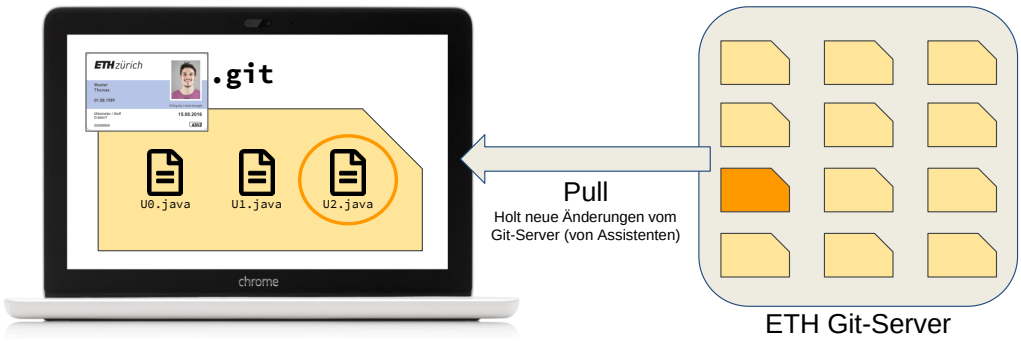
EBNF.txt

Was ist Git?

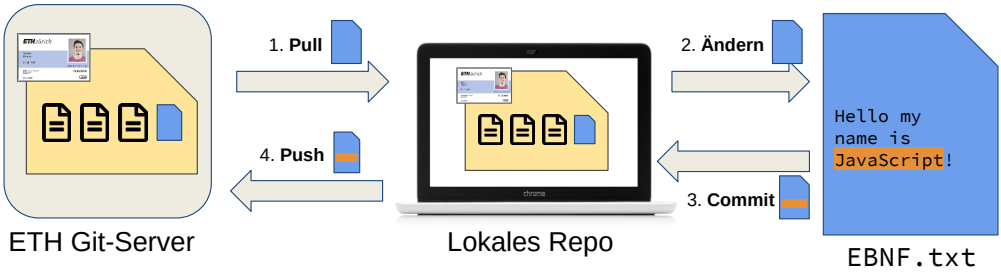
Git Push: Abgeben



Git Pull: Aufgaben / Feedback laden

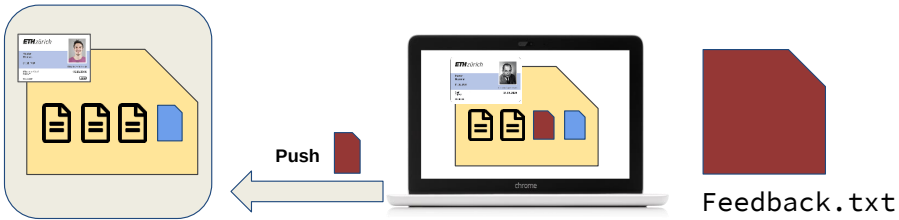


Git Pull/Push-Workflow



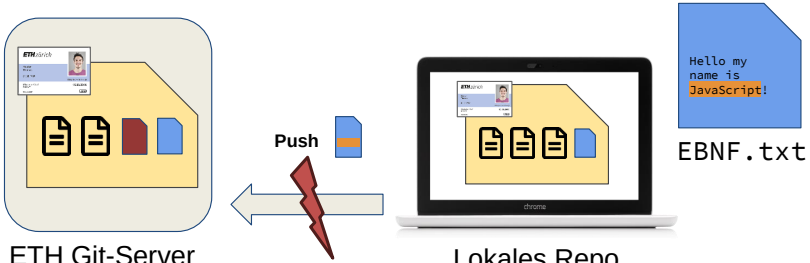
Was ist Git?

Git Repository ändert sich!



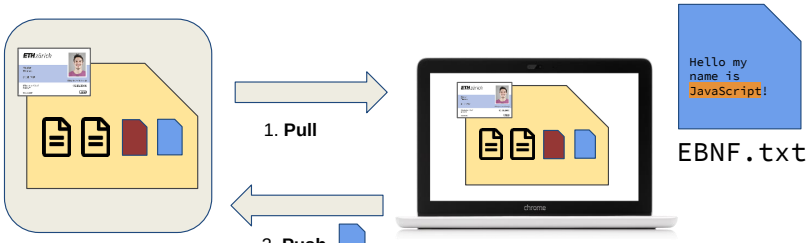
Was ist Git?

Git Merge Conflict



Was ist Git?

Git Merge Conflict verhindern



Immer zuerst pull dann push!

Demo

- Demo

6. EBNF Kahoot

Kahoot

7. Outro

Fragen/Unklarheiten?

- Nutzt das Studycenter für Eprog
- Wenn ihr möchtet das ich bestimmte Aufgaben nochmal bespreche dann schreibt mir gerne eure Wünsche
- Gerade kann alles noch etwas überwältigend sein, aber bleibt dran, ihr schafft das!

- That’s it

Extra

- Erstellen Sie eine Beschreibung <geradezahl>, die als legale Symbole alle geraden Zahlen (d.h. Zahlen, die ohne Rest durch 2 teilbar sind) zulässt. Beispiele sind +02, 4, 10, -20.

Extra

- Erstellen Sie eine Beschreibung <x2ygemischt>, die als legale Symbole genau jene Wörter zulässt, in denen für jedes "X" zwei "Y" als Paar auftreten. Beispiele sind XYY, YYX, XYYYYX, XXYYYY.