University of Basel

Versionsverwaltung mit Git

Cristina Illi <cristina.illi@unibas.ch>
Rahel Arnold <rahel.arnold@unibas.ch>
Simon Peterhans <simon.peterhans@unibas.ch>
Nikodem Kernbach <nikodem.kernbach@unibas.ch>
Jan Schönholz <jan.schoenholz@unibas.ch>

Departement Mathematik & Informatik, Universität Basel

Programmierprojekt – FS2020

Diese Woche: Git, IntelliJ

Voraussetzungen

- Java 11, git, IDE installiert
- Eingeloggt auf Scicore, Zugriff auf euer Gruppenrepo

Lernziele

- Basics von git (add, commit, push, pull) auf der Kommandozeile
- Mergekonflikte auf der Kommandozeile
- IntelliJ Basics (Projekt öffnen, Programm ausführen)

Was ist neu im Programmier-Projekt?

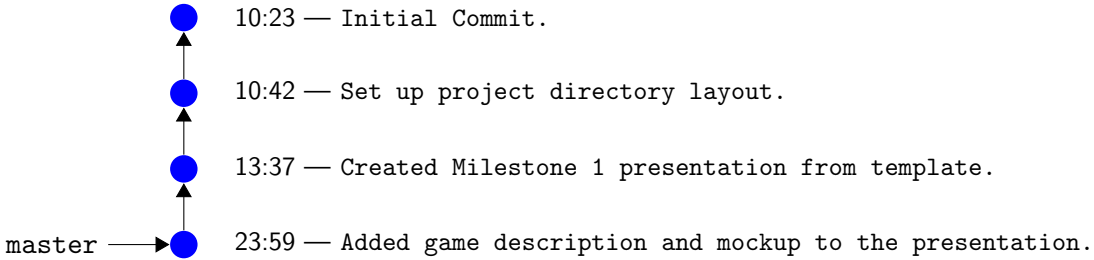
- Ihr seid nicht mehr **alleine** am Entwickeln.
- Wie **synchronisiert** man die Daten?
- Wie vermeidet man **Datenverlust**?
- Wie werden **Änderungen nachverfolgt**?

Die Lösung

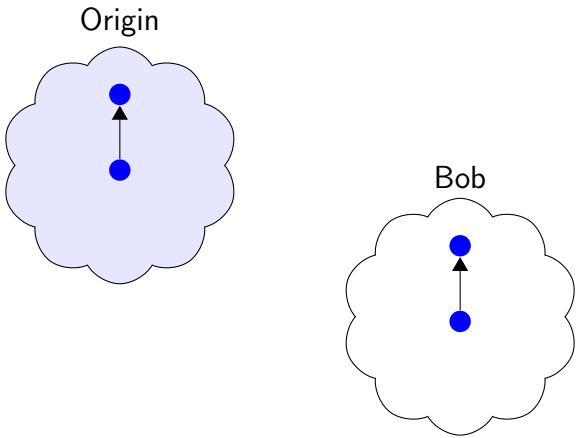
Terminologie: Git versus Gitlab / Github

- *git* ist eine in C geschriebene **Applikation** für **verteilte Systemverwaltung** (`https://github.com/git/git`)
- *Github* und *Gitlab* sind Firmen / Applikationen, welche **Infrastruktur & User Interfaces** für git zur Verfügung stellen.
- *SmartGit, GitKraken, SourceTree, TortoiseGit* sind alles **graphische Benutzeroberflächen** zu git.

Theorie: Versionsverwaltung

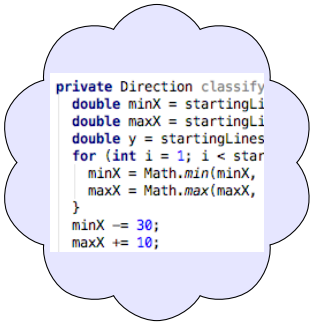


Theorie: **Verteilte** Versionsverwaltung

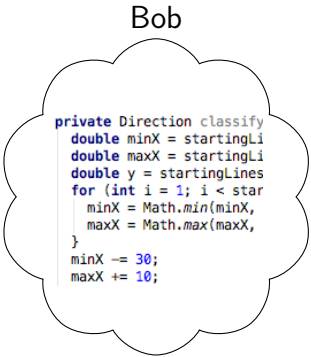


Theorie: Verteilte Versionsverwaltung

Origin



Bob



Praxis: Konfiguration

Ziel: Neue Entwicklerin (Alice) steigt in das Projekt ein

Konfiguration

```
> git config --global user.name "Dein Name"
> git config --global user.email "Unibas Mail"
```

Allgemeine Konfiguration eures Userprofils, damit git eure Commits zuordnen kann.

Klonen

```
> git clone <Repo-Url>
```

Repository klonen.

Theorie: Verändern

Origin

```
private Direction classify
  double minX = startingLi
  double maxX = startingLi
  double y = startingLines
  for (int i = 1; i < star
    minX = Math.min(minX,
    maxX = Math.max(maxX,
  }
  minX -= 30;
  maxX += 10;
```

Alice

```
private Direction classify
  double minX = startingLi
  double maxX = startingLi
  double y = startingLines
  for (int i = 1; i < star
    minX = Math.min(minX,
    maxX = Math.max(maxX,
  }
  minX -= 30;
  maxX += 40;
```

Bob

```
private Direction classify
  double minX = startingLi
  double maxX = startingLi
  double y = startingLines
  for (int i = 1; i < star
    minX = Math.min(minX,
    maxX = Math.max(maxX,
  }
  minX -= 30;
  maxX += 10;
```

Praxis: Verändern

Status sehen

```
› git status
  On branch master
  Your branch is up to date with 'origin/master'.

  Changes not staged for commit:
    modified: ../../../../../math/Direction.java
```

Wir haben lokale Änderungen, die wir noch nicht dem Repository hinzugefügt haben.

Praxis: Verändern

Verändern

- `git add <Dateiname>` (1)
- `git add <Ordnername>` (2)
- `git add .` (3)

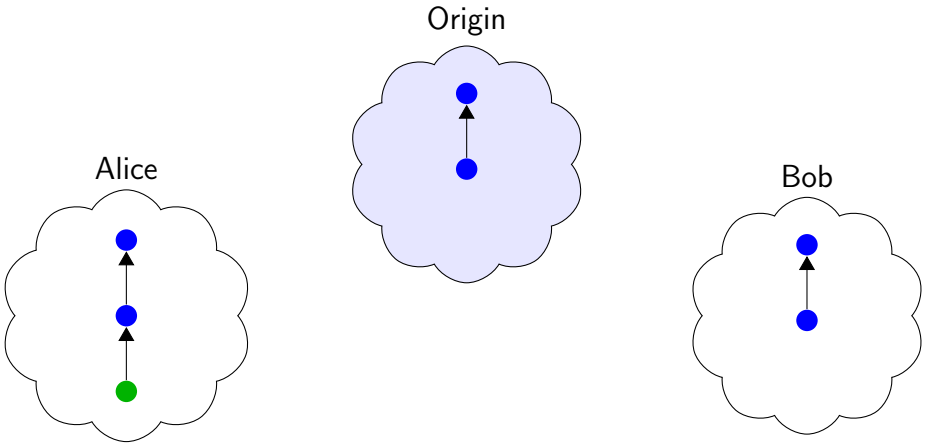
Fügt bestimmte Änderung (1), alle Änderungen in einem Ordner (2) oder alle Änderungen (3) dem **lokalen** Repository hinzu.

Committen

- `git commit -m "<Nachricht>"`

Committet alle **hinzugefügten** Änderungen (lokal).

Theorie: Verändern



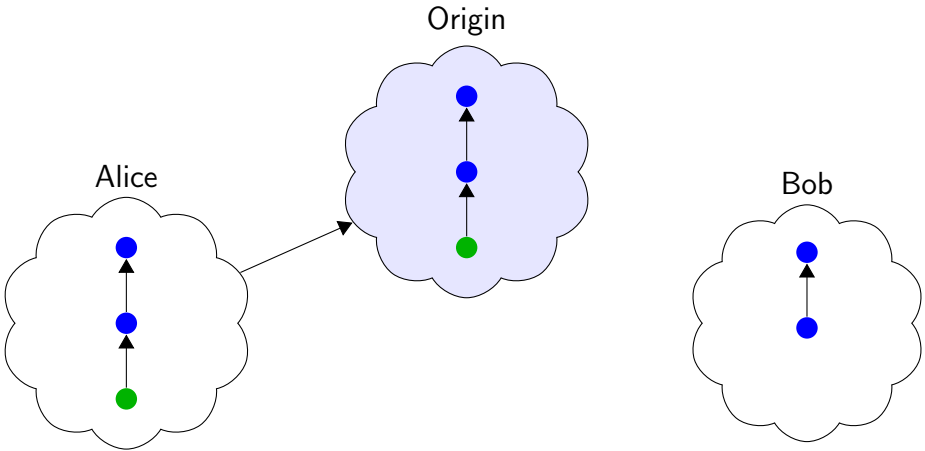
Praxis: Pushen

Pushen

```
> git push origin master
```

Pusht alle lokalen Änderungen auf `origin` in den Branch `master`.

Theorie: Pushen



Praxis: Pullen

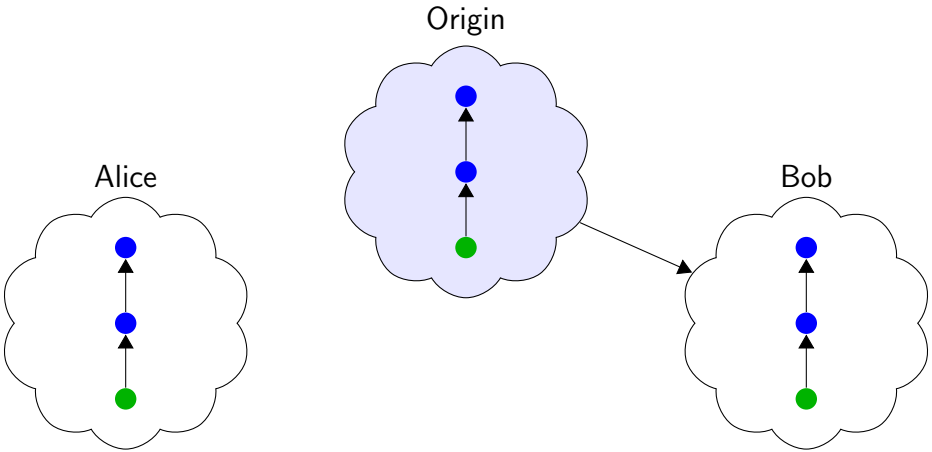
Ziel: Bob erhält die neuen Änderungen vom Server

Pullen

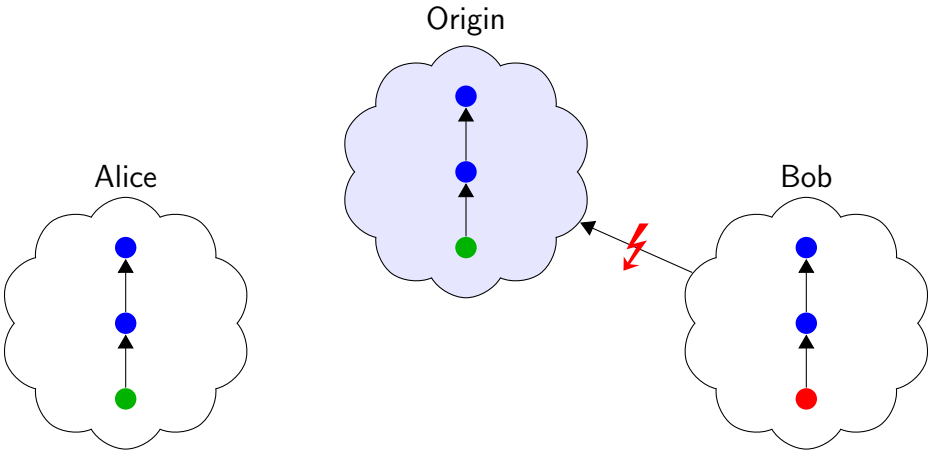
```
› git pull origin master
```

Alle Änderungen von `origin` **herunterladen** und mit den lokalen Änderungen **mergen**.

Theorie: Pullen



Theorie: Konfliktlösung



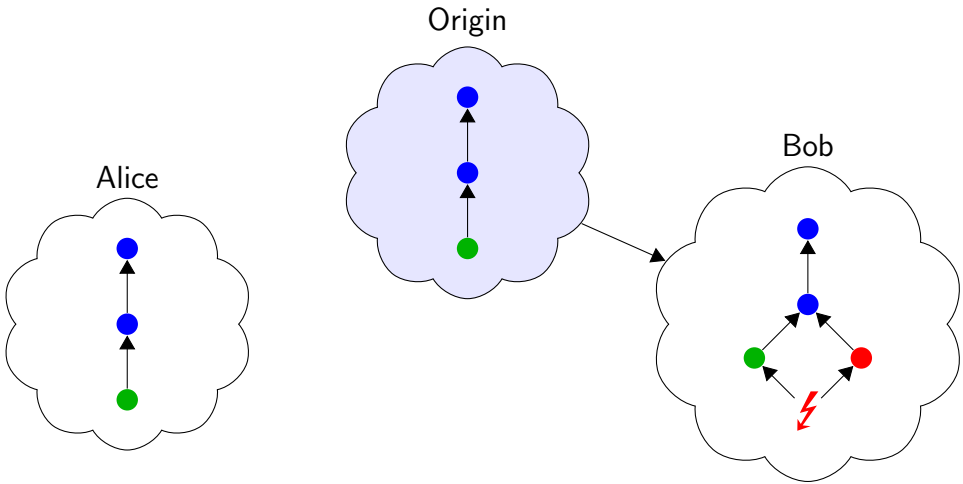
Praxis: Konfliktlösung

Push-Konflikt

```
> git push origin master
   ! [rejected] master -> master (fetch first)
  error: failed to push some refs to 'origin'
```

`origin` hat Änderungen, die lokal noch nicht vorhanden sind.

Theorie: Konfliktlösung



Praxis: Konfliktlösung

Merge-Konflikt

```
> git pull origin master
  From origin
    master     -> origin/master
  CONFLICT (content): Merge conflict in AUTHORS
  Automatic merge failed; fix conflicts and then commit the result.
```

`origin` hat Änderungen, die nicht automatisch mit den lokalen Änderungen vereint werden könnten.

Praxis: Konfliktlösung

Merge-Konflikt

```
<<<<<<< HEAD
Bob
=======
Alice
>>>>>>> 436ee81
```

Oben sind die lokalen Änderungen, unten die Änderungen von `origin`.

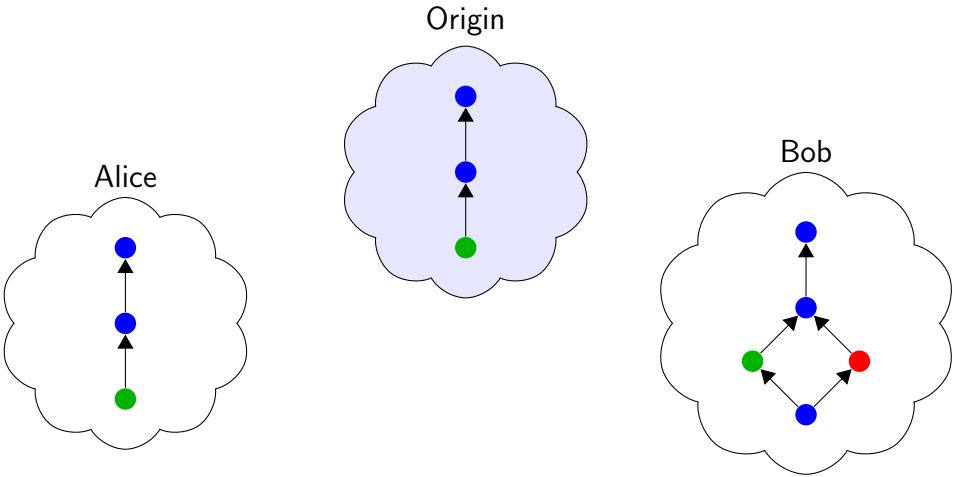
Praxis: Konfliktlösung

Merge-Konflikt

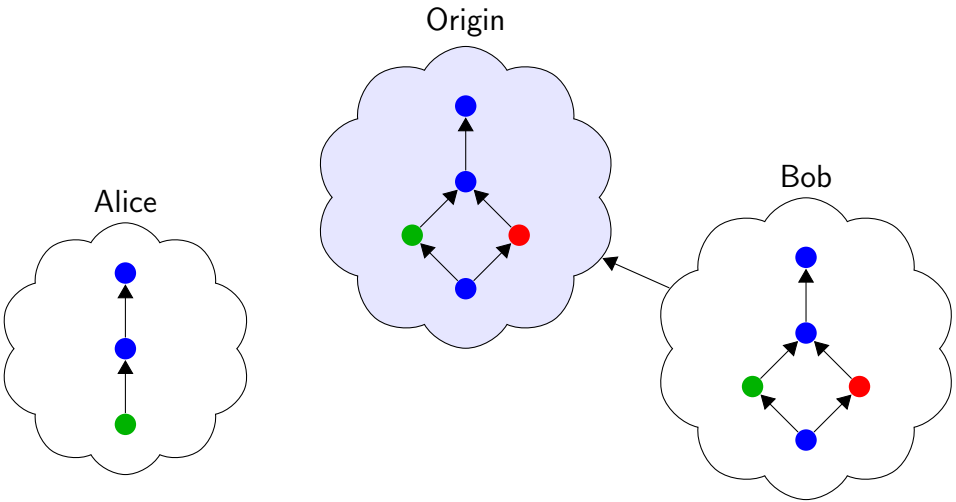
Nach erfolgreichem lösen des Konflikt (Kommandozeile, Text-Editor oder IDE)

```
git add .
git commit -m "Resolved merge conflict"
git pull
git push
```

Theorie: Konfliktlösung



Theorie: Konfliktlösung



Git: Best Practices

- **Oft committen** - Dem Server tut das nicht weh!
- Genau **eine** funktionale Änderung **pro** Commit
- Nur **funktionierenden** Code committen
- Jeweils **vor** dem Pushen updaten (`pull`)
- Ein .gitignore-file schützt euer repo vor Müll (e.g.: `http://gitignore.io/`)
- **Sinnvolle** Commit-Messages schreiben
  → Änderungen können einfacher nachvollzogen werden
  → sind üblicherweise auf Englisch

Gut

```
Fixed #11 - Added try catch block to prevent uncaught exception.
```

Schlecht

```
Uploaded file xyz
```

Git: Cheatsheet

- Traditionelles Cheatsheet:
  `https://www.git-tower.com/blog/git-cheat-sheet/`
- Umfangreiche textuelle Übersicht:
  Englisch:
  `https://services.github.com/on-demand/downloads/github-git-cheat-sheet/`
  Deutsch:
  `https://services.github.com/on-demand/downloads/de/github-git-cheat-sheet/`

Praxis: Git (10 minuten)

Jede_r auf seinem Rechner

- Repository der Gruppe clonen
- File mit eigenem Namen (e.g. bob.txt) pushen
- Den eigenen Namen in das File CONTRIBUTORS.txt schreiben (filenamen gross) & pushen
- Ziel: Pro Mitglied der Gruppe 1 File mit namen im Repo + 1 File CONTRIBUTORS.txt mit allen Namen im Repo
- Go!

Homework

- Alle mind. einen commit
- .gitignore konfigurieren für Gruppe

Fragen?