University of Basel

Versionsverwaltung mit Git

Silvan Heller <silvan.heller@unibas.ch>
Slides für CS108: Marcel Neidinger <m.neidinger@unibas.ch>

Department Mathematik & Informatik, Universität Basel

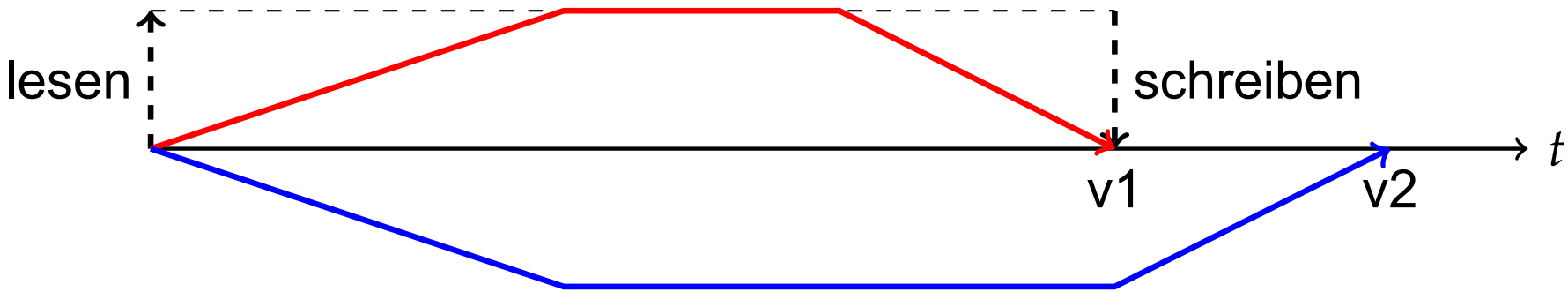
HS17– Software Engineering

Recap: Programmier-Projekt?

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

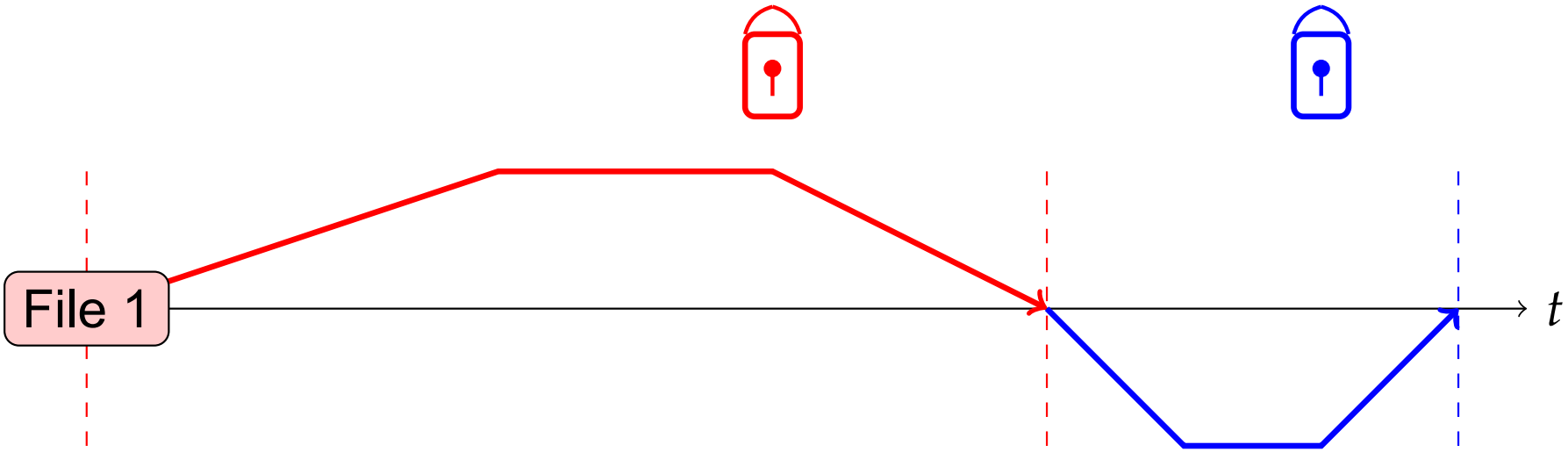
Dateiserver

- *Dateiserver* oder „Dropbox“



- Änderungen aus **v1** gehen verloren!

Verbesserung: Blockierender Dateiserver



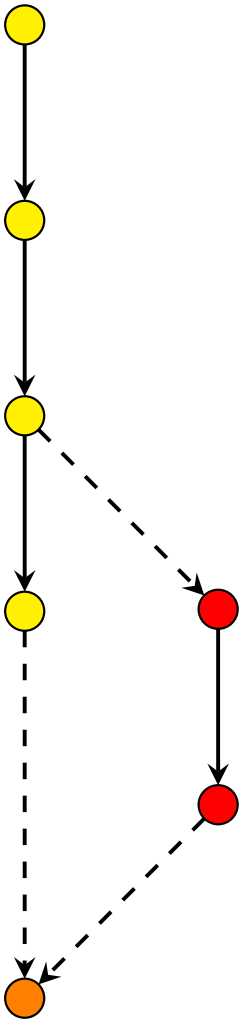
- **Lock-modify-unlock**
- Verzögerungen: gleichzeitiges Arbeiten nicht möglich
- Administrativer Aufwand: Was wenn vergessen wird zu entsperren?
- Äbhängigkeiten von Quelltextdateien werden ausser Acht gelassen

Die Lösung

Git

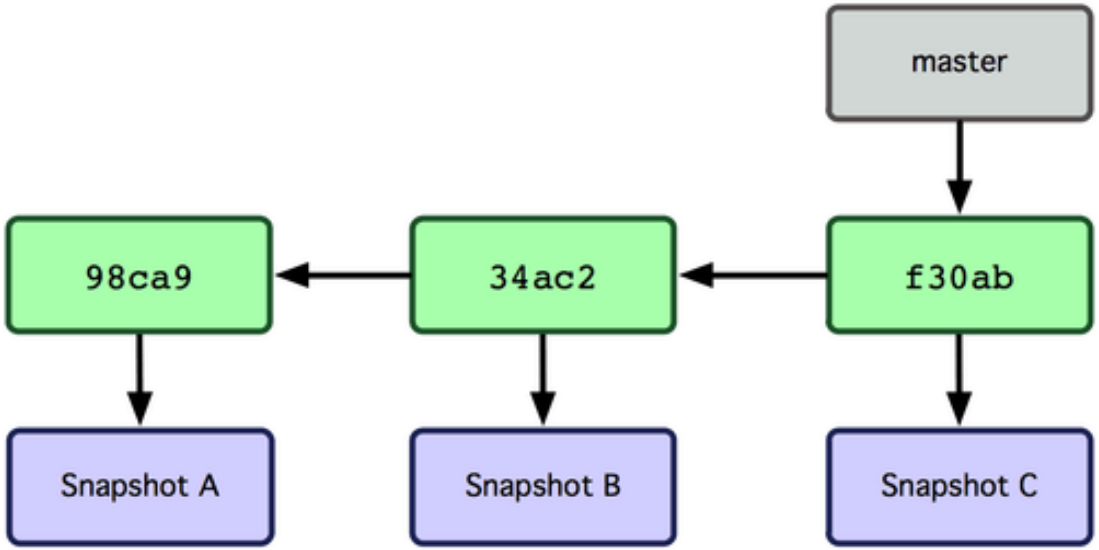

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Theorie: Das Baummodell

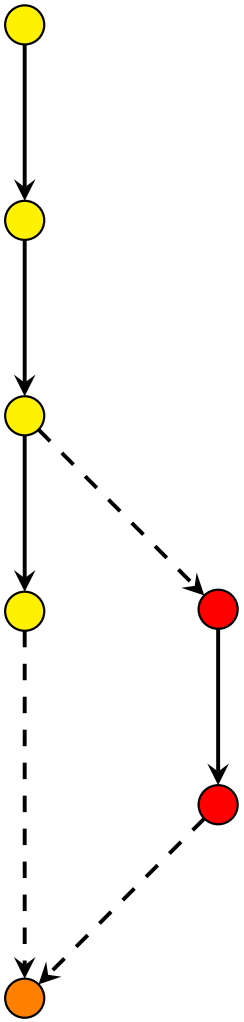


Commit Status (Änderungen) eures Codes

- Jeder **Commit** hat eindeutigen Hash

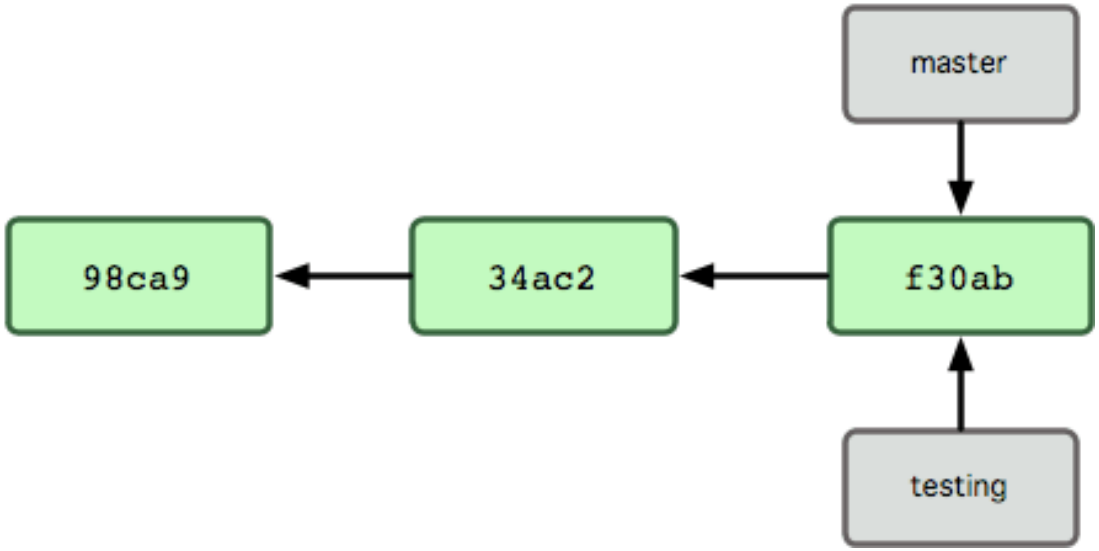


Theorie: Das Baummodell

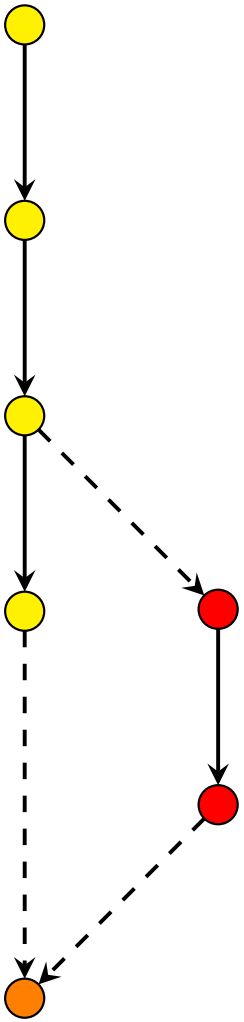


Commit Status (Änderungen) eures Codes

- Jeder **Commit** hat eindeutigen Hash

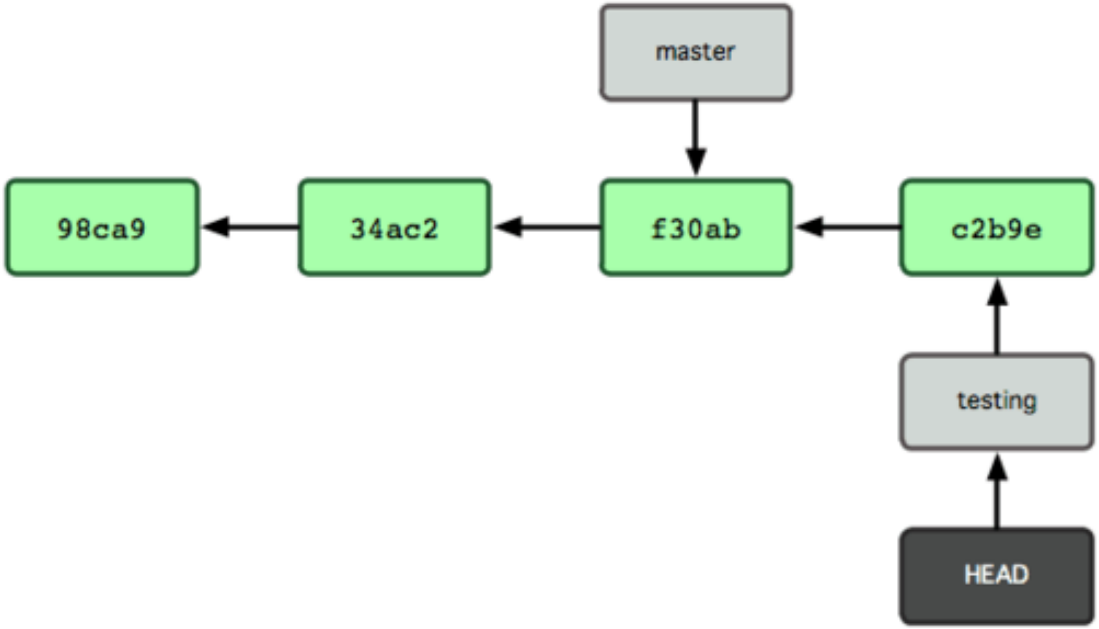


Theorie: Das Baummodell

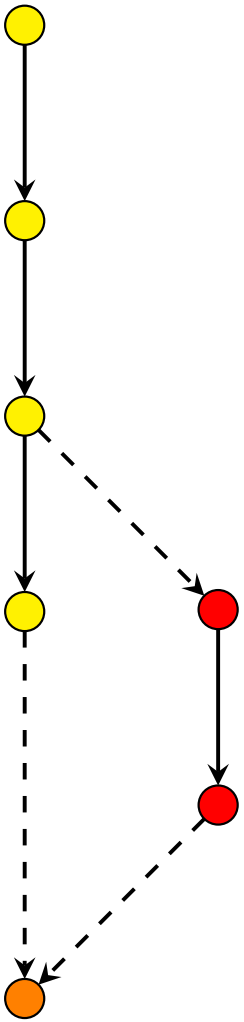


Commit Status (Änderungen) eures Codes

- Jeder **Commit** hat eindeutigen Hash

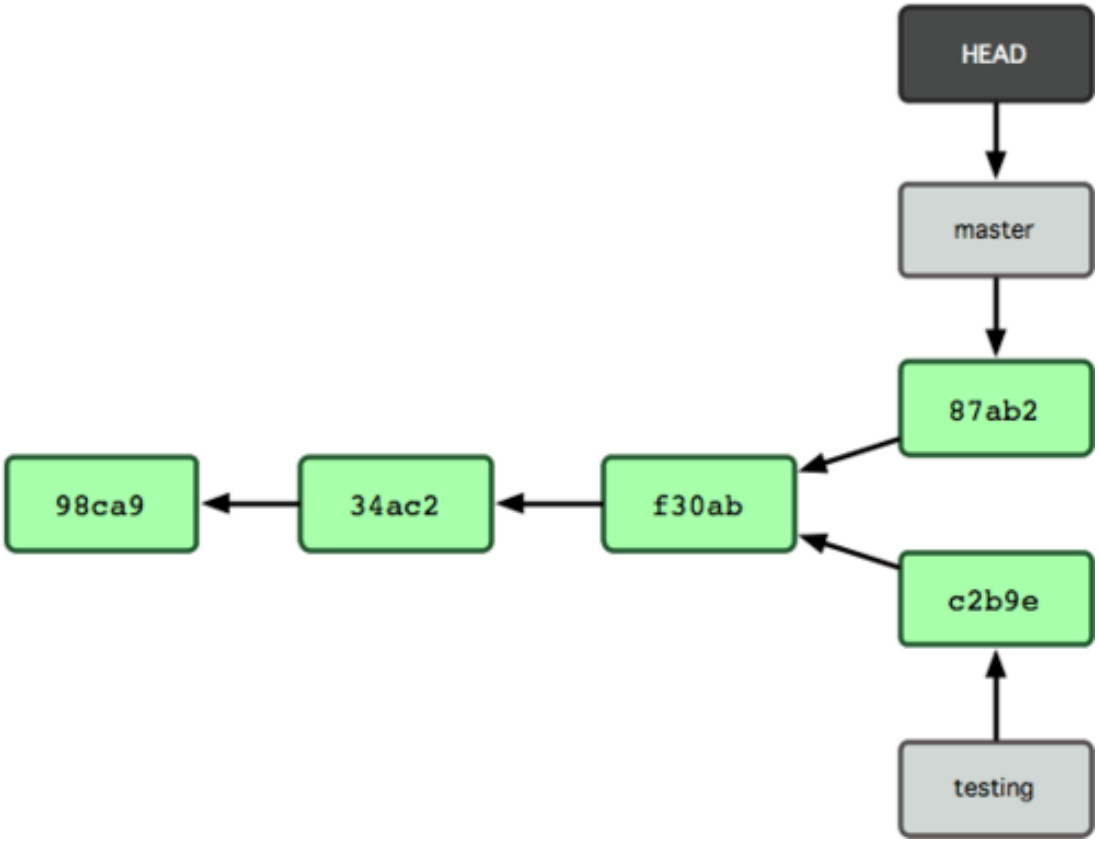


Theorie: Das Baummodell

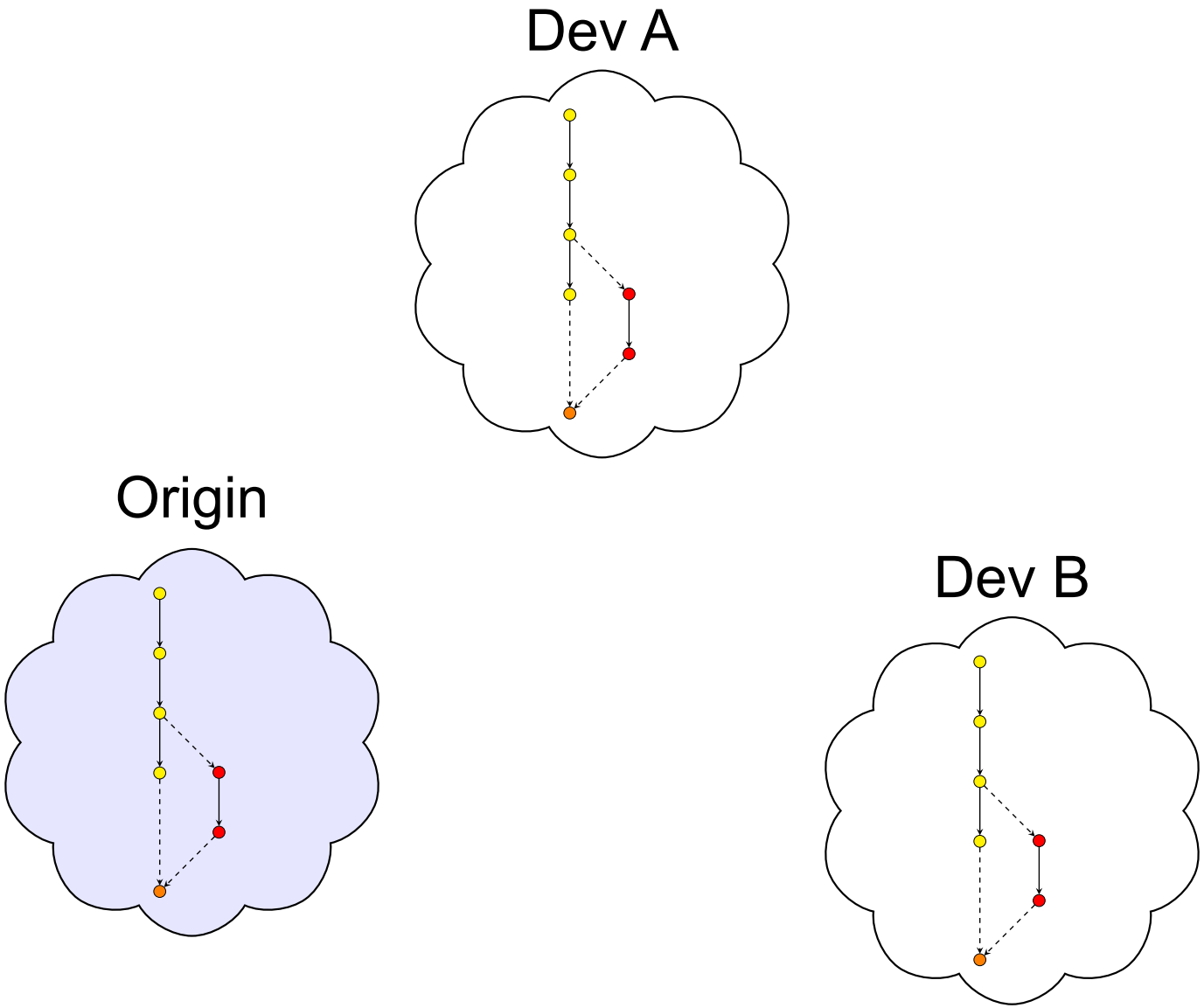


Commit Status (Änderungen) eures Codes

- Jeder **Commit** hat eindeutigen Hash



Theorie: Verteilte Versionsverwaltung



Theorie: Branchingmodel

Praxis: Konfiguration

Konfiguration

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

Allgemeine Konfiguration eures Userprofils, damit Github / Gitlab Eure commits zuordnen kann.

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, welche Infrastruktur & User Interfaces für git zur Verfügung stellen.
- *SmartGit, GitKraken, SourceTree, TortoiseGit* sind alles auch UIs um die Kommandozeile zu verwenden

Praxis: Klonen und Verändern

Klonen

- `git clone <Repo Url>`

Repository klonen $\rightarrow$herunterladen

Verändern

- `git add <Dateiname>`
- `git add .`

Fügt bestimmte Änderung oder alle Änderungen dem (lokalen!) Repository hinzu

Praxis: Commiten und Übertragen

Commiten

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

Commitet alle änderungen (lokal!)

Pushen

```
git push origin master
```

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

Git in den Übungen

Commandline versus IntelliJ versus tool

- IntelliJ kann alles was Ihr braucht
- Ihr dürft aber auch etwas anderes benützen

Fork

- Ihr forkt das Ganttproject repo.
- Pro Übungsblatt ein branch
- Abgabe via Pull Request auf master

Git: Best Practices

- **Oft Committen** - Dem Server tut das nicht weh!
- Genau eine funktionale Änderung pro Commit
- Nur **funktionierenden** Code committen
- Sinnvolle Commit-Kommentare schreiben. Das vereinfacht es zu verstehen was geändert wurde.
- Jeweils **vor** dem Committen updaten
- ’Pull before you push’ - Also nicht wie bei Türen

Gut

```
Fehler #32 behoben - GUI stürzt ohne Port Paramter
nicht mehr mit Null-Pointer Exception ab
```

Schlecht

```
Zeugs gemacht - Dinge geändert
```

Git: Cheatsheet

- Textuelle Zusammenfassung: `http://gitref.org/basic/`
- Traditionelles Cheatsheet:
  `https://www.git-tower.com/blog/git-cheat-sheet/`

Fragen?