Einführung in das wissenschaftliche Arbeiten

Einheit VI:
Nützliche Helfer

Dr. Björn Eichmann

Bochum, 29. März 2019

Übersicht nützlicher Tools

- Literaturverwaltung
  - JabRef, Mendeley, Citavi, EndNote, Zotero,…
- Auslesen von Daten (aus einer Grafik)
  - Engauge-Digitizer
- Online LaTeX Editor
  - Overleaf (ShareLaTeX)
- Projektmanagmentsoftware (PMS)
  - Trello (Redmine, OpenProject,…)
- Verteilte Versionsverwaltung (DVCS)
  - Git/ Github (BitKeeper, Bazaar, GNU arch, Darcs, Fossil,…)

Literaturverwaltungsprogramme

Welches Programm ist das richtige **für mich**?

Viele Enscheidungsfragen lassen sich objektiv beantworten:

http://mediatum.ub.tum.de/doc/1316333/1316333.pdf

Entscheidungsfragen

- Benutzerfreundlichkeit
- Betriebssystem (Linux, Mac, Windows, mobile Version?)
- Hilfe (einführende Kurse, Ansprechpartner, etc.)
- Kompatibilität (z.B. mit anderen Textverarbeitungsprogrammen)
- Kosten
- Funktionalität
  - Suche in Katalogen und Datenbanken aus dem Programm heraus?
  - PDF-Bearbeitung?
  - Literaturverzeichnis erstellen?
  - Online-Zugriff auf meine Quellen?
  - LaTeX-Schnittstelle
- Arbeitsplatz (an unterschiedlichen/ fremden Rechnern?)

Vorhanden oder nicht?

	Lizenzmodell kommerziell	kostenlos	**Installation** lokal	webbasiert	**Betriebssystem** Windows	Mac OS	Linux
Citavi	●	●	●	●	●	●	●
Colwiz	●	●	●	●	●	●	●
EndNote	●	●	●	●	●	●	●
JabRef	●	●	●	●	●	●	●
Mendeley	●	●	●	●	●	●	●
Zotero	●	●	●	●	●	●	●
ZoteroBib	●	●	●	●	●	●	●

● vorhanden
● nicht vorhanden

Vorhanden oder nicht?

	Nutzung über mobile Endgeräte	Recherche in Datenbanken	Import von Dateien	Export aus Datenbanken	Übernahme aus Webseiten	weitere Importmöglichkeiten	Ergänzung der Metadaten	Verknüpfung von Referenzen	Dublettencheck	Ordner / Gruppen zur Strukturierung	Freigabe für Externe / Öffentlichkeit	gemeinsame Bearbeitung	Social-Software-Funktionalitäten	textunabhängige Bibliographien	Verknüpfung mit Textverarbeitungsprogramm	Aktualisierung von BibTeX-Dateien	intuitive Nutzeroberfläche
Citavi	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Colwiz	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
EndNote	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
JabRef	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Mendeley	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Zotero	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
ZoteroBib	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

● vorhanden
● nicht vorhanden

Stärken und Schwächen

	Stärken	Schwächen
Citavi	+ > neben kommerzieller auch kostenlose Version mit vollem Funktionsumfang (beschränkt auf 100 Titel pro Projekt) + > Suche nach Volltexten + > umfangreiche Wissensorganisation und Aufgabenplanung + > sehr intuitive Benutzeroberfläche + > gemeinsame Bearbeitung für Teams v.a.in der Cloud möglich, bei besonderen Sicherheitserfordernissen Gruppen über einen eigenen Datenbankserver und Citavi for DBServer + > umfangreiche Hilfeangebote und sehr gute persönliche Endnutzerbetreuung + > leicht bedienbares Word-Add-In mit nützlichen Funktionen + > PDF-Reader mit vielen Bearbeitungsfunktionen + > Zitationsstil-Finder bisher einzigartig	– > nur für Windows – > keine Web-Version
Colwiz	+ > plattformunabhängig und kostenlos + > PDF-Reader mit vielen Bearbeitungsfunktionen + > App für iOS und Android + > Kollaborationsmöglichkeiten: Drive, Gruppen, Profile	– > fehleranfällig (Synchronisation) – > Begrenzung der Referenzen auf 5000 – > kein OpenURL – > Probleme beim Import von Dateien (txt, XML, PDF, etc.) – > nur einfache Suchfunktion innerhalb der eigenen Datenbank – > Hilfeseite teilweise nicht aktuell
EndNote	+ > für Windows und Mac + > Webversion für Abonnenten des Web of Science und Besitzer von EndNote Desktop kostenlos + > Webversion als EndNote Basic auch kostenlos nutzbar + > Manuscript Matcher zur Suche nach geeigneten Zeitschriften für die eigentliche Veröffentlichung + > iPad-App + > automatisierter Import von PDFs möglich + > Suche in Metadaten, PDF-Volltext und PDF-Kommentaren möglich	– > nicht sehr intuitiv – > Webversion: nur bibliographische Angaben können geteilt werden, Dateien nicht – > Parallele Funktion zum Teilen von Gruppen und Libraries bleibt trotz erweiterter Möglichkeiten in EndNote Desktop, verwirrend

Stärken und Schwächen

	Stärken	Schwächen
JabRef	+ > Mit Java auf Windows, Mac und Linux nutzbar + > gute Zusammenarbeit mit verschiedenen LaTeX-Editoren + > Ranking-, Relevanz- und Prioritätsangaben möglich + > Open Source und kostenlos + > intutitive Bedienung + > Suche / Download von Volltexten	– > keine Social-Software-Funktionalität
Mendeley	+ > Einzelplatzversion und webbasiert + > plattformunabhängig bzw. für Windows, Mac, Linux + > kostenlos (bis 2 GB) + > Zusammenarbeit innerhalb von Gruppen sehr gut möglich + > ausgeprägte Social-Software-Funktionalitäten + > durch Synchronisierung überall Zugriff auf die Daten	– > ISBN-Lookup fehlt – > seit 2-3 Jahren keine Weiterentwicklung des Produktes – > der Web-Importer liefert nicht immer optimale Ergebnisse
Zotero	+ > Einzelplatzversion und webbasiert + > Firefox-Addon direkt im Browser integriert, wo Recherchen durchgeführt werden + > plattformunabhängig bzw. für Windows, Mac, Linux + > kostenlos und vertrauenswürdige Serverumgebung, da Open Source + > Social-Software-Funktionalitäten: Freigabe von Papern über Interessensgruppen, Gruppenarbeit + > Erstellen von eigenständigen, titelunabhängigen Notizen	– > keine Recherche in Datenbanken aus Zotero heraus – > keine Bearbeitung der Volltexte – > keine hierarchischen Verknüpfungen möglich – > nur einfache Gliederungs- und Sortiermöglichkeit
ZoteroBib	+ > webbasiert, dadurch plattformunabhängig + > kostenlos und OpenSource + > einfache, intuitive Oberfläche + > keine Anmeldung nötig + > Teamfunktion + > gute Anzeige auf Mobilgeräten	– > geringer Funktionsumfang – > kein Plugin für Textverarbeitung – > bei Speicherung im Browser Verlust der Literaturliste möglich durch z.B. Löschen der Cookies – > keine umfangreiche Zugriffskontrolle bei Online-Speicherung - wer den Link kennt, kann die Liste bearbeiten

…der passende Anwender?

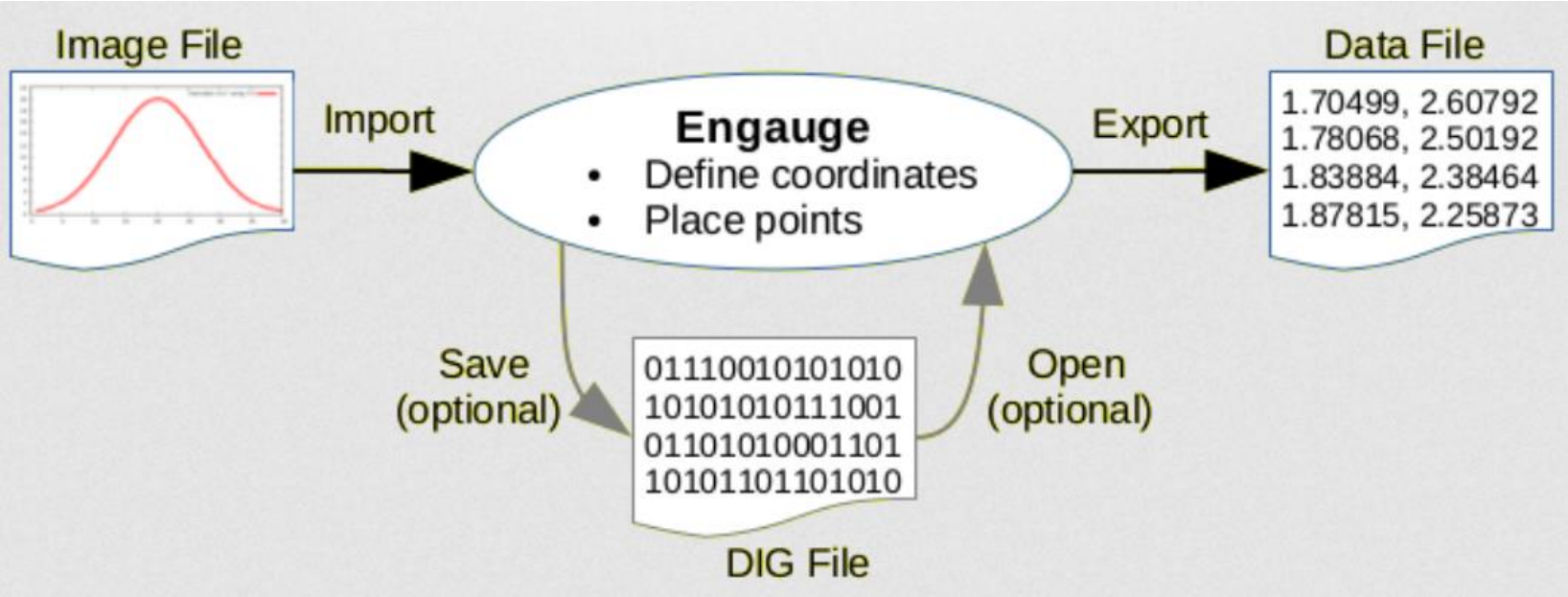
	Anwender *Für welchen Nutzer ist dieses Literaturverwaltungsprogramm geeignet? Einsteiger / Wissenschaftler / …?*
Citavi	> Windows-User > intuitive Nutzung vom Einsteiger bis zum Profi
Colwiz	> Programm durch Fehleranfälligkeit und nicht-intuitive Oberfläche im Vergleich zu kostenpflichten Angeboten nicht zu empfehlen
EndNote	> für Windows- und Mac-User geeignet mit entsprechender Einarbeitungszeit für alle Nutzergruppen geeignet

	Anwender *Für welchen Nutzer ist dieses Literaturverwaltungsprogramm geeignet? Einsteiger / Wissenschaftler / …?*
JabRef	> besonders für LaTeX-User geeignet > auch für sehr große Literatursammlungen > fachlich (Datenbank-Auswahl) besonders für MINT-Bereich > sowohl für Einsteiger als auch erfahrene Benutzer geeignet
Mendeley	> für alle Nutzergruppen geeignet > vor allem für Nutzer geeignet, die (fast) nur mit PDF-Volltexten arbeiten
Zotero	> für Studienanfänger, da kostenlos > besser für kleinere Literatursammlungen > alle Forschungsbereiche
ZoteroBib	> für Einsteiger geeignet, die keine Textverarbeitung benötigen > für alle geeignet, die schnell und einfach reine Literaturlisten bearbeiten wollen oder die Literaturliste mit anderen Programmen (z. B. LaTeX) weiterverarbeiten möchten

Engauge-Digitizer

- Open-Source Programm
  - Linux, Mac, Windows
  - http://markummitchell.github.io/engauge-digitizer/

- Das Gegenteil eines Grafik-Tools:



Einsatzbereich

- Dargestellte Daten nicht als Zahlenwerte verfügbar,
  - weil Altdaten verloren (gilt es tunlichst zu vermeiden!)
  - weil Fremdautoren diese nicht angeben
    - überprüfe auch den Anhang und das ergänzende Material („supplementary material“)

- Alternatives Vorgehen
  - (Fremd-)Autoren kontaktieren
  - Kollegen fragen
  - Datenbank aufsuchen
    - Cosmic Ray Database: https://lpsc.in2p3.fr/cosmic-rays-db/
    - NASA/IPAC Extragalactic Database: https://ned.ipac.caltech.edu/

Online-LaTeX Editor

Overleaf (+ShareLaTeX):

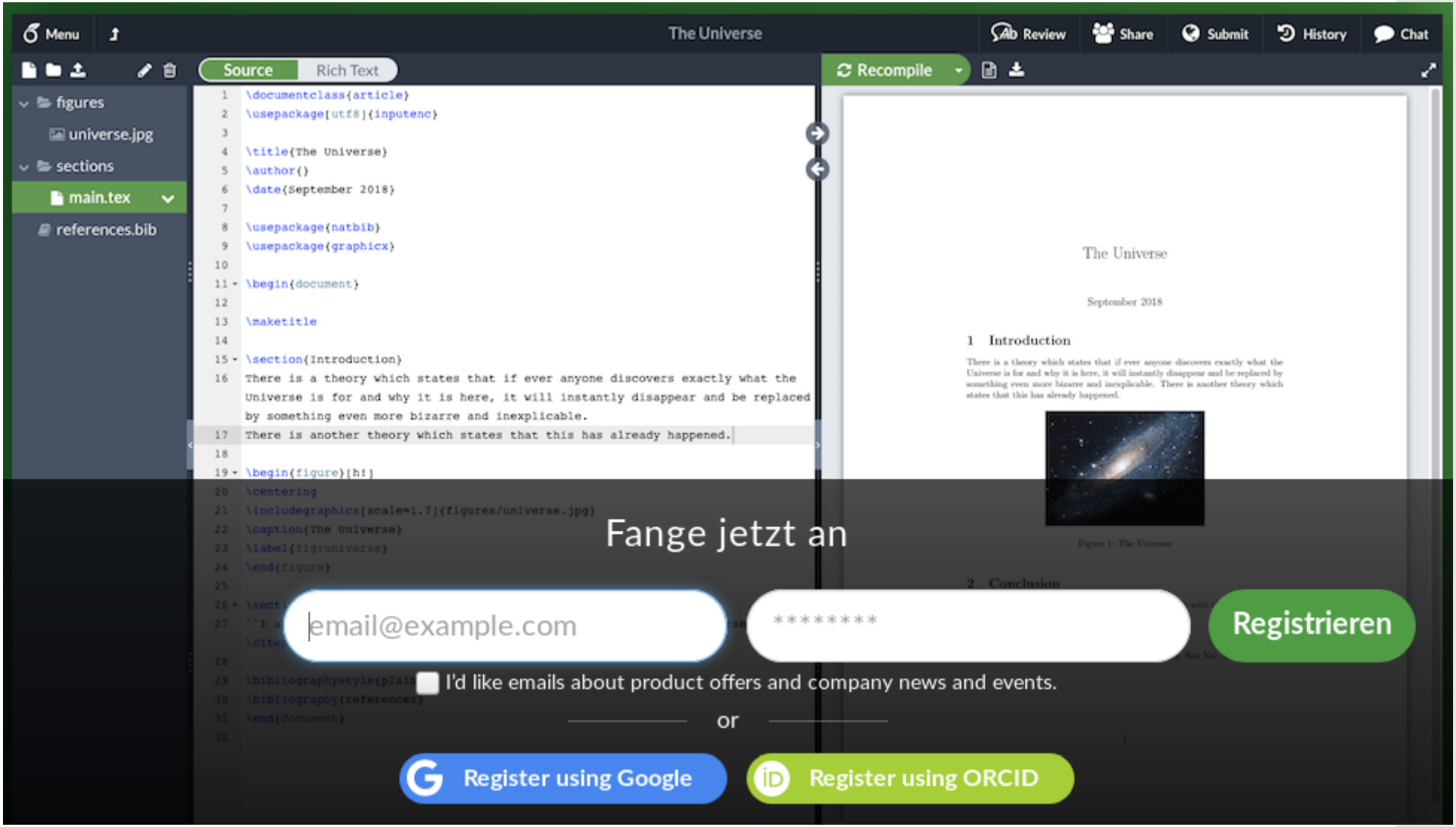
- Freemium, web-basierte Anwendung
- Echtzeit-Kollaboration
- direktes Kompilieren eines PDF
- Rechtschreibprüfung
- Vielzahl an Templates
- Offline arbeiten durch Synchronisation mit Github oder Dropbox

Overleaf

https://de.overleaf.com/

RUB

Overleaf

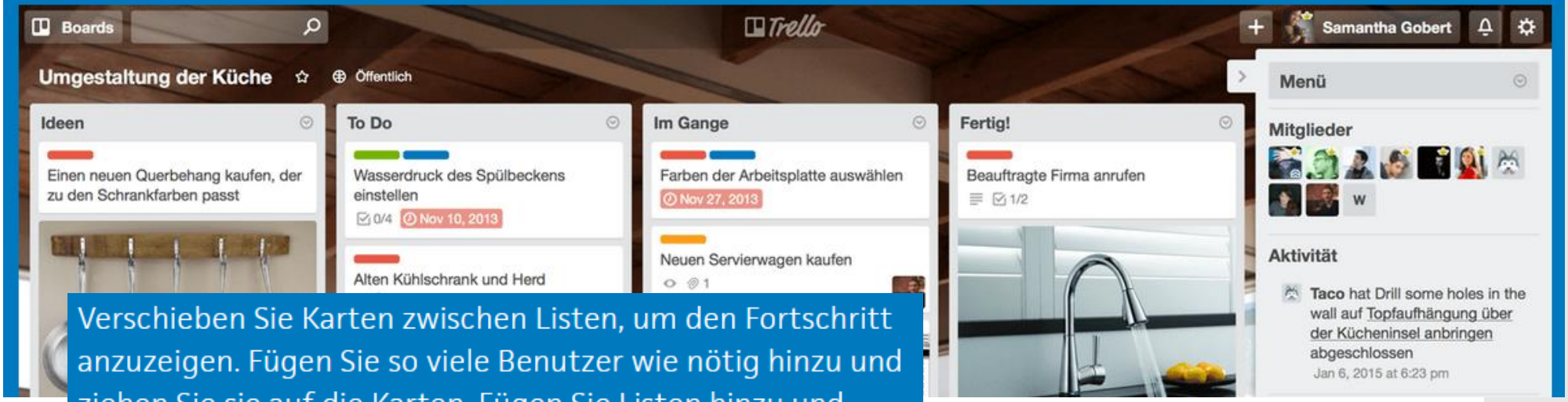


Projektmanagmentsoftware (PMS)

- Unterstützung bei der Durchführung von Projekten:
  - Terminplanung,
  - Ideengenerierung,
  - Projektplanung,
  - Aufgabenmanagment,
  - Dokumentation von Ergebnissen, u.ä.
- Insbesondere bei hoher Projektkomplexität und vielen Beteiligten sinnvoll
- z.T. branchenspezifisch, unterschiedliche Funktionalität
- kostenfreie Software: Trello, Redmine, OpenProjekt

Trello

Dies ist ein Trello-Board. Es handelt sich um eine Liste von Listen, die mit Karten gefüllt sind. Diese werden mit einem Team oder von Ihnen genutzt.



Verschieben Sie Karten zwischen Listen, um den Fortschritt anzuzeigen. Fügen Sie so viele Benutzer wie nötig hinzu und ziehen Sie sie auf die Karten. Fügen Sie Listen hinzu und organisieren Sie sie wie benötigt. **Trello passt sich Ihrem Projekt, Team und Arbeitsablauf an.**

Sie sehen alle Informationen Ihres Projektes mit einem Blick auf das Board. Alles wird in Echtzeit aktualisiert. Sie müssen nichts einrichten und alle Benutzer begreifen es sofort.

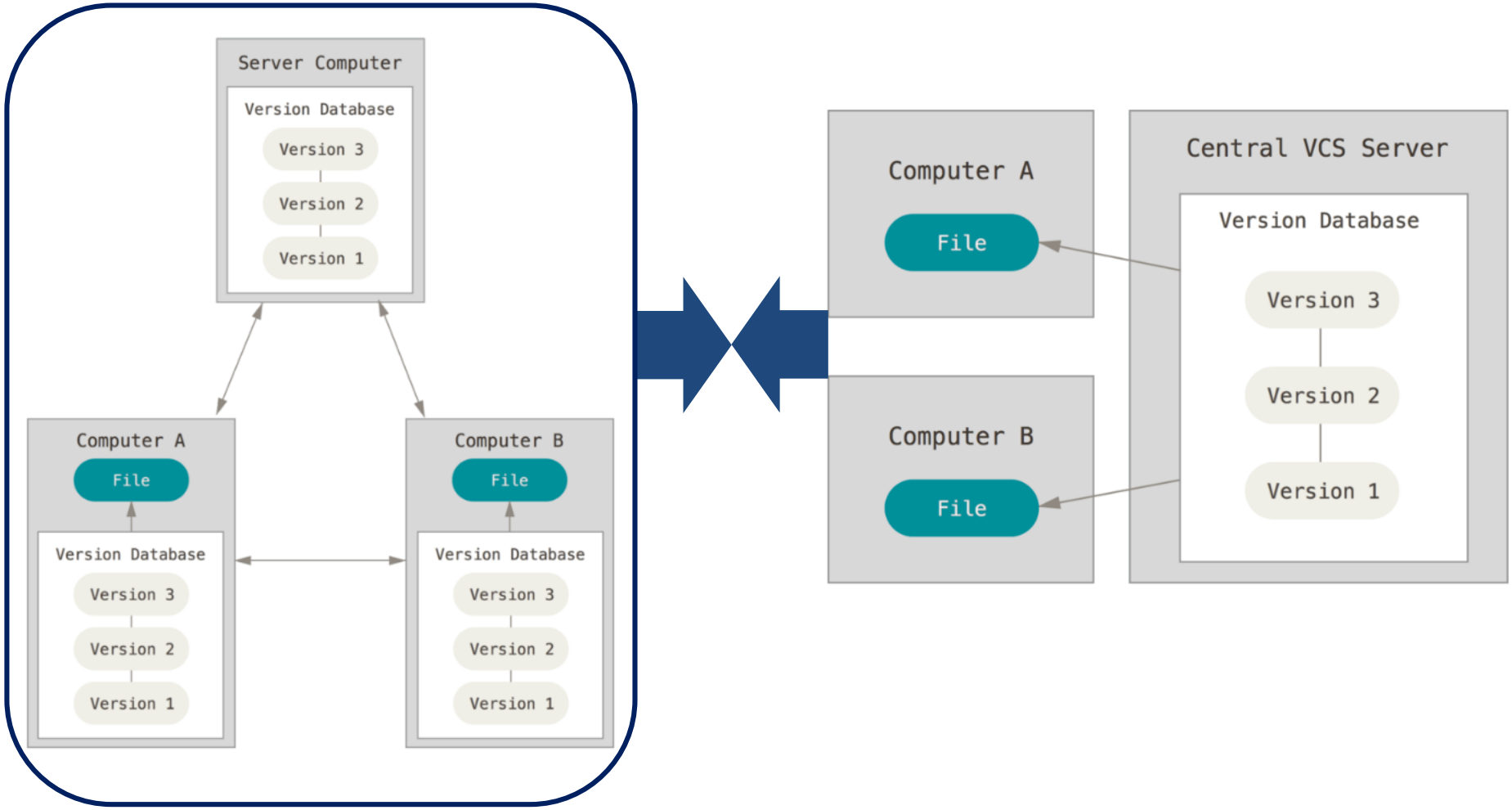
Trello

https://trello.com/

RUB

Verteilte Versionsverwaltung (DVCS)

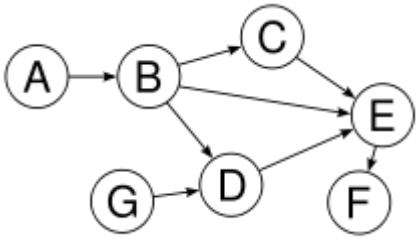
- kein zentrales Repository (→← zentrale Versionsverwaltung)



Verteilte Versionsverwaltung (DVCS)

- kein zentrales Repository (→← zentrale Versionsverwaltung)
- **jeder hat sein eigenes *Repository*** und kann dieses mit jedem beliebigen anderen Repository abgleichen

➢ verteilte Versionsgeschichte



- weitgehend automatische ***Merge-Mechanismen*** ermöglichen es unterschiedliche, parallel entwickelte Versionen zusammenzuführen
- Üblicherweise existiert ein ***offizielles Repository***, das von neuen Projektbeteiligten zu Beginn ihrer Arbeit ***geklont***, d.h. auf das lokale System kopiert, wird.

Git

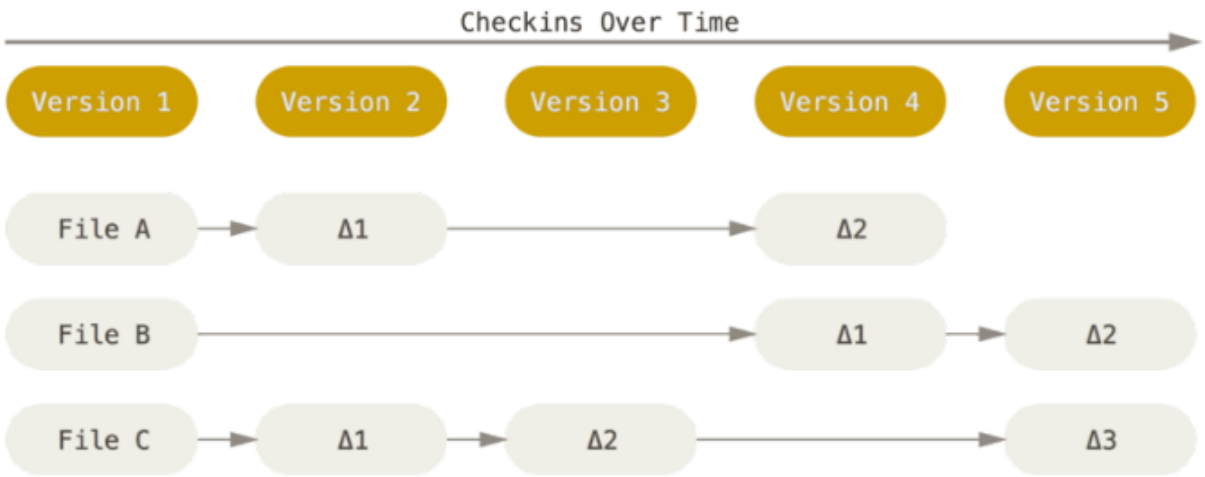
- **Freie, open-source** Software zur verteilten Versionsverwaltung von Dateien
- Von **Linus Torvalds** (Linux) 2005 initiiert, nach Lizenzänderungen des bis dahin kostenlosen BitKeeper-Systems
- Zentrale Anforderungen:
  - Unterstützung verteilter Arbeitsabläufe
  - Sehr hohe Sicherheit gegen Verfälschungen (SHA-1)
  - Hohe Effizienz (bei großen Projekten)
  - Einfaches Design/ Handling

https://git-scm.com/

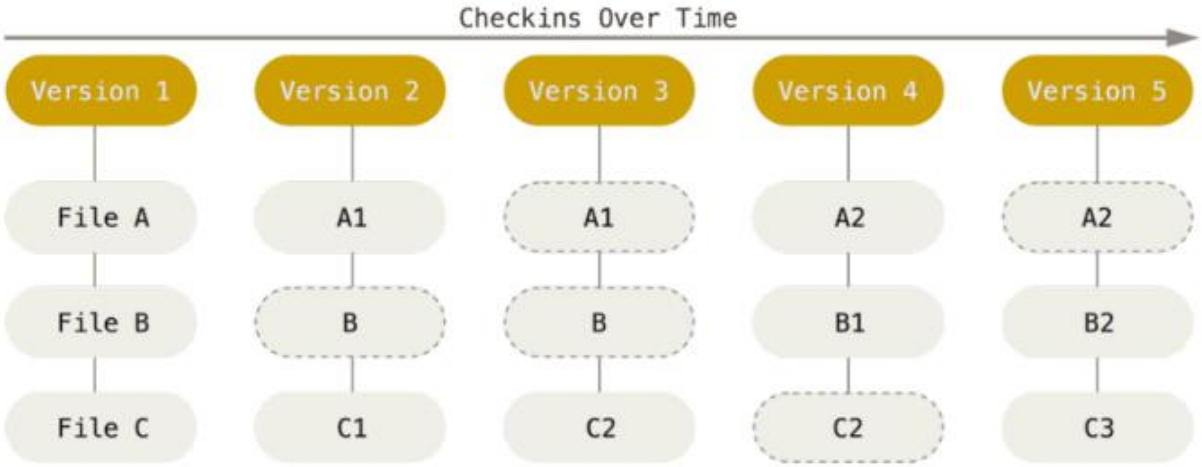
Snapshots, not differences!

Speichern von Informationen:

- Basierend auf den Änderungen („andere“ VCS)



- als „Stream of Snapshots“ (Git)
  unveränderte Files werden nicht erneut gespeichert, nur ein Link zur letzten Version



Grundbegriffe

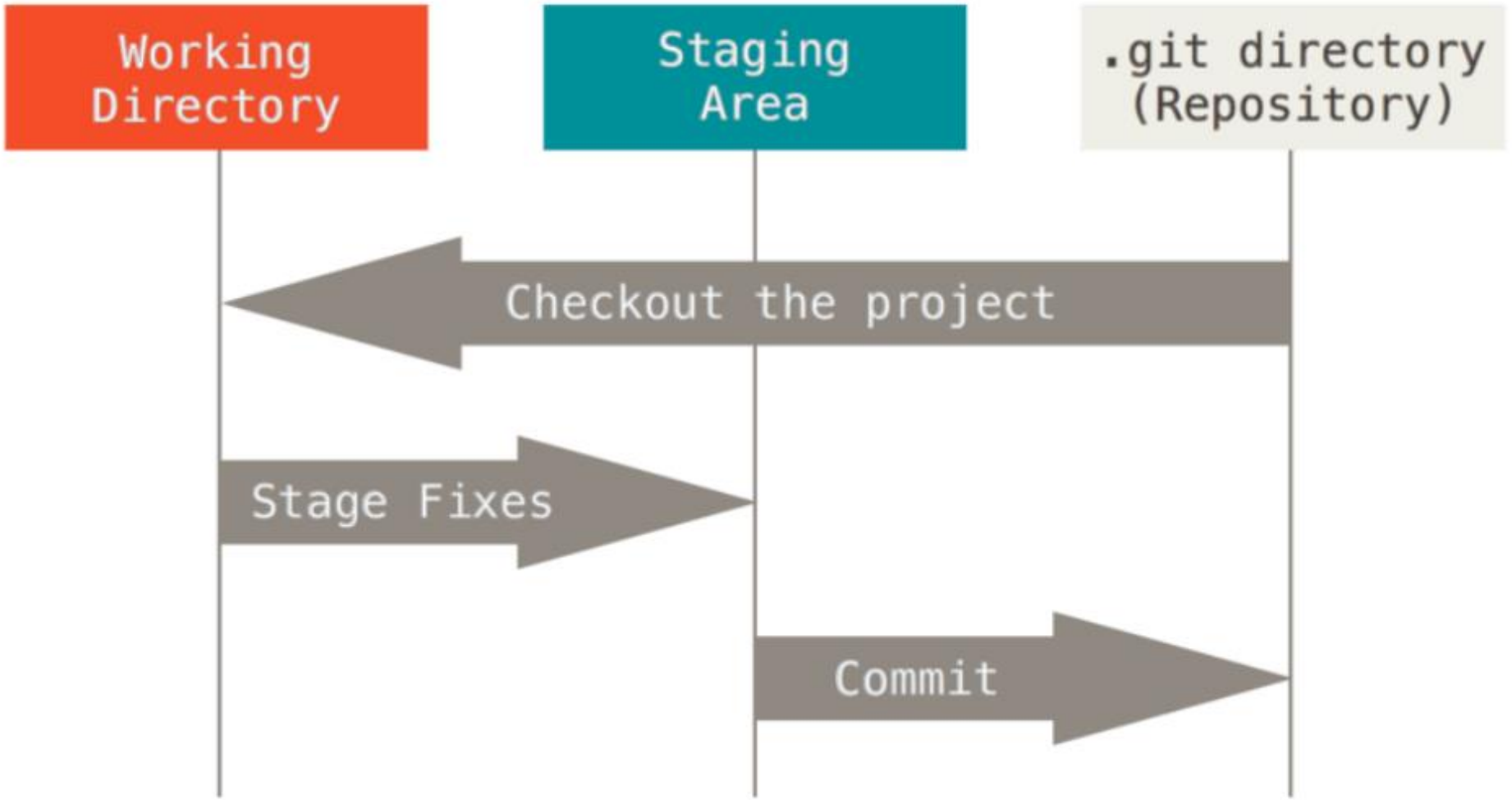
- *Repository*
  - Datenbank, in der die Zustände jeder Datei abgelegt werden
- *Working Tree (Working Directory/ Working Files)*
  - Arbeitsverzeichnis, in dem die Datei modifiziert wird
- *Commit*
  - Veränderungen am *Working Tree,* sowie Metadaten (Autor, Datum, Uhrzeit, Nachricht) die diese beschreibt, werden im *Repository* gespeichert
  - referenziert immer den Zustand *aller* verwalteten Dateien
- *Index (Stage/ Cache)*
  - *Indiziert,* welche Änderungen an welchen Dateien eines *Commits*

Grundbegriffe

- *Clone*
  - Klon eines Git-Repositories
- *Branch*
  - Abzweigung in der Entwicklung, um bspw. neue Features zu entwickeln
- *Merge*
  - Zusammenführen von *Branches*
- master
  - Ausgangsbranch, der beim Initialisieren eines neuen Repositories erstellt wird (Git braucht mind. einen Branch)
  - Kein technischer Unterschied zu anderen *Branches*

RUB

Drei Zustände: *comitted, modified, staged*



Erste Schritte

Git installieren:

- Linux: **$ sudo apt install git-all**
  - For more information: http://git-scm.com/download/linux
- Mac: **$ git –version**

  (falls Git nicht bereits installiert ist, werden sie zur Installation aufgefordert)
- Windows (Git for Windows): https://git-scm.com/download/win
  - For more information: https://git-for-windows.github.io/

Hilfe:

- Zu einem beliebigen Git Kommando: **$ git help <verb>**
- Zu den verfügbaren Optionen eines Git Kommandos: **--help** or **-h**
- Einführung und Handbuch: https://git-scm.com/doc

RUB

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
eiche@localhost:~$ cd GitProjekt_test/
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

nichts zu committen (Erstellen/Kopieren Sie Dateien und benutzen Sie "git add" zum Versionieren)
eiche@localhost:~/GitProjekt_test$ geany &
[1] 4714
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
[1]+  Fertig                  geany
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        hello.py

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren)
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
eiche@localhost:~$ cd GitProjekt_test/
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

nichts zu committen (Erstellen/Kopieren Sie Dateien und benutzen Sie "git add" zum Versionieren)
eiche@localhost:~/GitProjekt_test$ geany &
[1] 4714
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
[1]+  Fertig                  geany
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        hello.py

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren)
```

Konfiguration des Nutzers

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
eiche@localhost:~$ cd GitProjekt_test/
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

nichts zu committen (Erstellen/Kopieren Sie Dateien und benutzen Sie "git add" zum Versionieren)
eiche@localhost:~/GitProjekt_test$ geany &
[1] 4714
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
[1]+  Fertig                  geany
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        hello.py

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren)
```

Repository erstellen

RUB

Erste Schritte – ein Beispiel ~~(lokal)~~

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
```

Repository erstellen

Achtung: Falls das Repository nicht lokal erzeugt wird sondern **auf einem Server**, besser die folgenden *optionalen Parameter* mitgeben:

$ git init --bare --shared

erlaubt Nutzern in das (nicht-lokale) Repository zu *pushen*

blankes Repository (ohne Arbeitsverzeichnis)

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
eiche@localhost:~$ cd GitProjekt_test/
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

nichts zu committen (Erstellen/Kopieren Sie Dateien und benutzen Sie "git add" zum Versionieren)
eiche@localhost:~/GitProjekt_test$ geany &
[1] 4714
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
[1]+  Fertig                  geany
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        hello.py

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren)
```

Aktueller Zustand im Arbeitsverzeichnis:
Wir stehen vor dem ersten Commit, aber nichts zum commiten gefunden

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
eiche@localhost:~$ cd GitProjekt_test/
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

nichts zu committen (Erstellen/Kopieren Sie Dateien und benutzen Sie "git add" zum Versionieren)
eiche@localhost:~/GitProjekt_test$ geany &
[1] 4714
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
[1]+  Fertig                  geany
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        hello.py

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren)
```

Python Datei
<<hello.py>>
erstellen & testen

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~$ git config --global user.name "Björn Eichmann"
eiche@localhost:~$ git config --global user.email "eiche@tp4.rub.de"
eiche@localhost:~$ git init GitProjekt_test
Leeres Git-Repository in /home/eiche/GitProjekt_test/.git/ initialisiert
eiche@localhost:~$ cd GitProjekt_test/
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

nichts zu committen (Erstellen/Kopieren Sie Dateien und benutzen Sie "git add" zum Versionieren)
eiche@localhost:~/GitProjekt_test$ geany &
[1] 4714
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
[1]+  Fertig                  geany
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        hello.py

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie "git add" zum Versionieren)
```

Aktueller Zustand im Arbeitsverzeichnis :

Git registriert, dass sich Dateien in diesem Verzeichnis befinden, diese aber unversioniert (untracked), d.h. nicht bekannt, sind.

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git add hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)

        neue Datei:     hello.py

eiche@localhost:~/GitProjekt_test$ git commit -m "Hello die Erste"
[master (Basis-Commit) 77e2b98] Hello die Erste
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master
nichts zu committen, Arbeitsverzeichnis unverändert
```

Datei der *Staging Areal* dem *Index* hinzugefügt

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git add hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)

        neue Datei:     hello.py

eiche@localhost:~/GitProjekt_test$ git commit -m "Hello die Erste"
[master (Basis-Commit) 77e2b98] Hello die Erste
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master
nichts zu committen, Arbeitsverzeichnis unverändert
```

Aktueller Zustand im Arbeitsverzeichnis :

Git registriert, dass sich Dateien in diesem Verzeichnis befinden, welche zum commiten vorbereitet sind.

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git add hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)

        neue Datei:     hello.py

eiche@localhost:~/GitProjekt_test$ git commit -m "Hello die Erste"
[master (Basis-Commit) 77e2b98] Hello die Erste
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master
nichts zu committen, Arbeitsverzeichnis unverändert
```

Option -m übergibt eine Commit-Nachricht

Commit ins Repository

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git add hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master

Initialer Commit

zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)

        neue Datei:     hello.py

eiche@localhost:~/GitProjekt_test$ git commit -m "Hello die Erste"
[master (Basis-Commit) 77e2b98] Hello die Erste
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master
nichts zu committen, Arbeitsverzeichnis unverändert
```

Aktueller Zustand im Arbeitsverzeichnis: Sauber

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ geany &
[1] 6610
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master
Änderungen, die nicht zum Commit vorgemerkt sind:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
  (benutzen Sie "git checkout -- <Datei>...", um die Änderungen im Arbeitsverzeichnis zu verwerfen)

        geändert:       hello.py

keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git commit -a")
eiche@localhost:~/GitProjekt_test$ git add hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "Hello die Zweite"
[master d89a76b] Hello die Zweite
 1 file changed, 2 insertions(+)
eiche@localhost:~/GitProjekt_test$ git log --oneline
d89a76b Hello die Zweite
77e2b98 Hello die Erste
```

- Datei abändern/erweitern;
- *Index* hinzufügen;
- ins *Repository* commiten.

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ geany &
[1] 6610
eiche@localhost:~/GitProjekt_test$ git status
Auf Branch master
Änderungen, die nicht zum Commit vorgemerkt sind:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
  (benutzen Sie "git checkout -- <Datei>...", um die Änderungen im Arbeitsverzeichnis zu verwerfen)

        geändert:       hello.py

keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git commit -a")
eiche@localhost:~/GitProjekt_test$ git add hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "Hello die Zweite"
[master d89a76b] Hello die Zweite
 1 file changed, 2 insertions(+)
eiche@localhost:~/GitProjekt_test$ git log --oneline
d89a76b Hello die Zweite
77e2b98 Hello die Erste
```

Commit-ID und erste Zeile der Beschreibung

Projektgeschichte untersuchen

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
eiche@localhost:~/GitProjekt_test$ git checkout 77e2b98
Hinweis: Checke '77e2b98' aus.

Sie befinden sich im Zustand eines 'lösgelösten HEAD'. Sie können sich
umschauen, experimentelle Änderungen vornehmen und diese committen, und
Sie können alle möglichen Commits, die Sie in diesem Zustand machen,
ohne Auswirkungen auf irgendeinen Branch verwerfen, indem Sie einen
weiteren Checkout durchführen.

Wenn Sie einen neuen Branch erstellen möchten, um Ihre erstellten Commits
zu behalten, können Sie das (jetzt oder später) durch einen weiteren Checkout
mit der Option -b tun. Beispiel:

  git checkout -b <neuer-Branchname>

HEAD ist jetzt bei 77e2b98... Hello die Erste
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
eiche@localhost:~/GitProjekt_test$ git checkout -b "Hello_GanzGenau"
Zu neuem Branch 'Hello_GanzGenau' gewechselt
```

Die 2. Version von <<hello.py>>

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
eiche@localhost:~/GitProjekt_test$ git checkout 77e2b98
Hinweis: Checke '77e2b98' aus.

Sie befinden sich im Zustand eines 'lösgelösten HEAD'. Sie können sich
umschauen, experimentelle Änderungen vornehmen und diese committen, und
Sie können alle möglichen Commits, die Sie in diesem Zustand machen,
ohne Auswirkungen auf irgendeinen Branch verwerfen, indem Sie einen
weiteren Checkout durchführen.

Wenn Sie einen neuen Branch erstellen möchten, um Ihre erstellten Commits
zu behalten, können Sie das (jetzt oder später) durch einen weiteren Checkout
mit der Option -b tun. Beispiel:

  git checkout -b <neuer-Branchname>

HEAD ist jetzt bei 77e2b98... Hello die Erste
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
eiche@localhost:~/GitProjekt_test$ git checkout -b "Hello_GanzGenau"
Zu neuem Branch 'Hello_GanzGenau' gewechselt
```

Früheren Zustand wiederherstellen (alternativ: **git reset**)

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
eiche@localhost:~/GitProjekt_test$ git checkout 77e2b98
Hinweis: Checke '77e2b98' aus.

Sie befinden sich im Zustand eines 'lösgelösten HEAD'. Sie können sich
umschauen, experimentelle Änderungen vornehmen und diese committen, und
Sie können alle möglichen Commits, die Sie in diesem Zustand machen,
ohne Auswirkungen auf irgendeinen Branch verwerfen, indem Sie einen
weiteren Checkout durchführen.

Wenn Sie einen neuen Branch erstellen möchten, um Ihre erstellten Commits
zu behalten, können Sie das (jetzt oder später) durch einen weiteren Checkout
mit der Option -b tun. Beispiel:

  git checkout -b <neuer-Branchname>

HEAD ist jetzt bei 77e2b98... Hello die Erste
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
eiche@localhost:~/GitProjekt_test$ git checkout -b "Hello_GanzGenau"
Zu neuem Branch 'Hello_GanzGenau' gewechselt
```

Die 1. Version von <<hello.py>>

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
eiche@localhost:~/GitProjekt_test$ git checkout 77e2b98
Hinweis: Checke '77e2b98' aus.

Sie befinden sich im Zustand eines 'lösgelösten HEAD'. Sie können sich
umschauen, experimentelle Änderungen vornehmen und diese committen, und
Sie können alle möglichen Commits, die Sie in diesem Zustand machen,
ohne Auswirkungen auf irgendeinen Branch verwerfen, indem Sie einen
weiteren Checkout durchführen.

Wenn Sie einen neuen Branch erstellen möchten, um Ihre erstellten Commits
zu behalten, können Sie das (jetzt oder später) durch einen weiteren Checkout
mit der Option -b tun. Beispiel:

  git checkout -b <neuer-Branchname>

HEAD ist jetzt bei 77e2b98... Hello die Erste
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
eiche@localhost:~/GitProjekt_test$ git checkout -b "Hello_GanzGenau"
Zu neuem Branch 'Hello_GanzGenau' gewechselt
```

Branch-Bezeichnung (Leerzeichen nicht erlaubt)

neuen *Branch* erstellt

RUB

Erste Schritte – ein Beispiel (lokal)

Wichtigsten Optionen beim Verwalten der *Branches:*

- **$ git branch**
  listet lokale Branches auf
- **$ git branch <branch>**
  erstellt einen neuen Branch <branch>
- **$ git branch -m <neuer_name>**
  aktuelle Branch wird in <neuer-name> umbenannt
- **$ git branch -d <branch>**
  löscht <branch>, falls dieser im aktuellen enthalten ist
- **$ git branch -D <branch>**
  löscht <branch>, auch wenn dieser nicht im aktuellen enthalten ist
  (Achtung: diese Commits können verlorengehen)

Branch-Bezeichnung (Leerzeichen nicht erlaubt)

```
eiche@localhost:~/GitProjekt_test$ git checkout -b "Hello_GanzGenau"
Zu neuem Branch 'Hello_GanzGenau' gewechselt
```

neuen *Branch* erstellt

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ geany &
[2] 6816
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello user
```

Neue Version von <<hello.py>> erstellen

```
eiche@localhost:~/GitProjekt_test$ git add --patch
diff --git a/hello.py b/hello.py
index 44159b3..7e615d8 100644
--- a/hello.py
+++ b/hello.py
@@ -1 +1,3 @@
+###With a detailed hello
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1 +1,2 @@
+###With a detailed hello
 print("Hello world")
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -1 +2,2 @@
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ geany &
[2] 6816
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello user
eiche@localhost:~/GitProjekt_test$ git add --patch
diff --git a/hello.py b/hello.py
index 44159b3..7e615d8 100644
--- a/hello.py
+++ b/hello.py
@@ -1 +1,3 @@
+###With a detailed hello
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1 +1,2 @@
+###With a detailed hello
 print("Hello world")
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -1 +2,2 @@
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

oder kurz: **-p**

Commit schrittweise erstellen

Diese Zeilen sind neu hinzugekommen

Erste Schritte – ein Beispiel (lokal)

```
Stage this hunk [y,n,q,a,d,/,e,?]? ?
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
```

```
eiche@localhost:~/GitProj
[2] 6816
eiche@localhost:~/GitProjek
Hello world
Hello user
eiche@localhost:~/Git
diff --git a/hello.py
index 44159b3..7e615d8
--- a/hello.py
+++ b/hello.py
@@ -1 +1,3 @@
+###With a detailed he
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1 +1,2 @@
+###With a detailed hello
 print("Hello world")
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -1 +2,2 @@
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ geany &
[2] 6816
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello user
eiche@localhost:~/GitProjekt_test$ git add --patch
diff --git a/hello.py b/hello.py
index 44159b3..7e615d8 100644
--- a/hello.py
+++ b/hello.py
@@ -1 +1,3 @@
+###With a detailed hello
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1 +1,2 @@
+###With a detailed hello
 print("Hello world")
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -1 +2,2 @@
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

Commit schrittweise erstellen

Diese Zeilen sind neu hinzugekommen

Hunk konnte erfolgreich geteilt werden

Hunk nicht übernehmen

Hunk übernehmen

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ geany &
[2] 6816
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello user
eiche@localhost:~/GitProjekt_test$ git add --patch
diff --git a/hello.py b/hello.py
index 44159b3..7e615d8 100644
--- a/hello.py
+++ b/hello.py
@@ -1 +1,3 @@
+###With a detailed hello
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 h
@@ -1 +1,2 @@
+###With a det
 print("Hello
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -1 +2,2 @@
 print("Hello world")
+print("Hello user")
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

Commit schrittweise erstellen

Hunks schrittwiese aus dem *Index* entfernen:

$ git reset -p

Erste Schritte – ein Beispiel (lokal)

Wenn wir nun wieder zum *master Branch* wechseln wollen

```
eiche@localhost:~/GitProjekt_test$ git checkout master
```

RUB

Erste Schritte – ein Beispiel (lokal)

Wenn wir nun wieder zum *master Branch* wechseln wollen

```
eiche@localhost:~/GitProjekt_test$ git checkout master

error: Ihre lokalen Änderungen in den folgenden Dateien würden beim Auschecken
überschrieben werden:
        hello.py
Bitte committen oder stashen Sie Ihre Änderungen, bevor Sie Branches
wechseln.
Abbruch
```

Achtung: Änderungen noch nicht *commitet, daher zunächst:*

```
eiche@localhost:~/GitProjekt_test$ git commit -m "Hello an den Nutzer"
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git checkout master
Zu Branch 'master' gewechselt
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
```

zum *master Branch* wechseln

```
eiche@localhost:~/GitProjekt_test$ git merge Hello_GanzGenau
automatischer Merge von hello.py
KONFLIKT (Inhalt): Merge-Konflikt in hello.py
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
eiche@localhost:~/GitProjekt_test$ geany hello.py &
[3] 7123
eiche@localhost:~/GitProjekt_test$ git add hello.py
[3]+  Fertig                    geany hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "unikaten print Befehle aus beiden branches übernommen"
[master 1f7cb07] unikaten print Befehle aus beiden branches übernommen
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git checkout master
Zu Branch 'master' gewechselt
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
```

dortige <<hello.py>> Script ausführen

```
eiche@localhost:~/GitProjekt_test$ git merge Hello_GanzGenau
automatischer Merge von hello.py
KONFLIKT (Inhalt): Merge-Konflikt in hello.py
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
eiche@localhost:~/GitProjekt_test$ geany hello.py &
[3] 7123
eiche@localhost:~/GitProjekt_test$ git add hello.py
[3]+  Fertig                    geany hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "unikaten print Befehle aus beiden branches übernommen"
[master 1f7cb07] unikaten print Befehle aus beiden branches übernommen
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git checkout master
Zu Branch 'master' gewechselt
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum

eiche@localhost:~/GitProjekt_test$ git merge Hello GanzGenau
automatischer Merge von hello.py
KONFLIKT (Inhalt): Merge-Konflikt in hello.py
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
eiche@localhost:~/GitProjekt_test$ geany hello.py &
[3] 7123
eiche@localhost:~/GitProjekt_test$ git add hello.py
[3]+  Fertig                    geany hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "unikaten print Befehle aus beiden branches übernommen"
[master 1f7cb07] unikaten print Befehle aus beiden branches übernommen
```

angegebene *Branch* wird in den *aktuell ausgecheckten Branch* (d.h. **HEAD**) integriert

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git checkout master
Zu Branch 'master' gewechselt
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
```

```
eiche@localhost:~/GitProjekt_test$ git merge Hello_GanzGenau
automatischer Merge von hello.py
KONFLIKT (Inhalt): Merge-Konflikt in hello.py
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
eiche@localhost:~/GitProjekt_test$ geany hello.py &
[3] 7123
eiche@localhost:~/GitProjekt_test$ git add hello.py
[3]+  Fertig                    geany hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "unikaten print Befehle aus beiden branches übernommen"
[master 1f7cb07] unikaten print Befehle aus beiden branches übernommen
```

Die zwei Versionen der Datei <<hello.py>> weisen unterschiedliche Versionen einer Zeile auf.

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git checkout master
Zu Branch 'master' gewechselt
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum

eiche@localhost:~/GitProjekt_test$ git merge Hello_GanzGe
automatischer Merge von hello.py
KONFLIKT (Inhalt): Merge-Konflikt in hello.py
Automatischer Merge fehlgeschlagen; beheben Sie die Konfl                    Ergebnis.
eiche@localhost:~/GitProjekt_test$ geany hello.py &
[3] 7123
eiche@localhost:~/GitProjekt_test$ git add hello.py
[3]+  Fertig                  geany hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "unikaten print Befehle aus beiden branches übernommen"
[master 1f7cb07] unikaten print Befehle aus beiden branche
```

hello.py

```
1  <<<<<<< HEAD
2  ###Python test document
3  print("Hello world")
4  print("Hello Bochum")
5  =======
6  ###With a detailed hello
7  print("Hello world")
8  print("Hello user")
9  >>>>>>> Hello_GanzGenau
10
```

hello.py

```
1  ###Python test document
2  print("Hello world")
3  print("Hello Bochum")
4  print("Hello user")
5
```

Erste Schritte – ein Beispiel (lokal)

```
eiche@localhost:~/GitProjekt_test$ git checkout master
Zu Branch 'master' gewechselt
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
```

```
eiche@localhost:~/GitProjekt_test$ git merge Hello_GanzGenau
automatischer Merge von hello.py
KONFLIKT (Inhalt): Merge-Konflikt in hello.py
Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen Sie dann das Ergebnis.
eiche@localhost:~/GitProjekt_test$ geany hello.py &
[3] 7123
eiche@localhost:~/GitProjekt_test$ git add hello.py
[3]+  Fertig                    geany hello.py
eiche@localhost:~/GitProjekt_test$ git commit -m "unikaten print Befehle aus beiden branches übernommen"
[master 1f7cb07] unikaten print Befehle aus beiden branches übernommen
```

- Änderungen per **git add** hinzufügen (ohne Konflikt-Marker sieht Git den Konflikt als gelöst);
- unter Erklärung wie der Konflikt gelöst wurde *commiten*

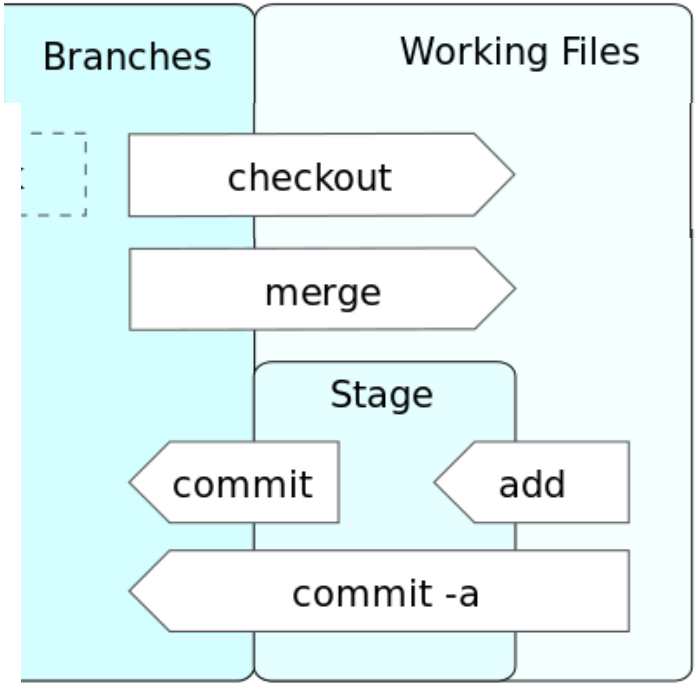
```
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Bochum
Hello user
```

Erste Schritte – ein Beispiel (lokal)

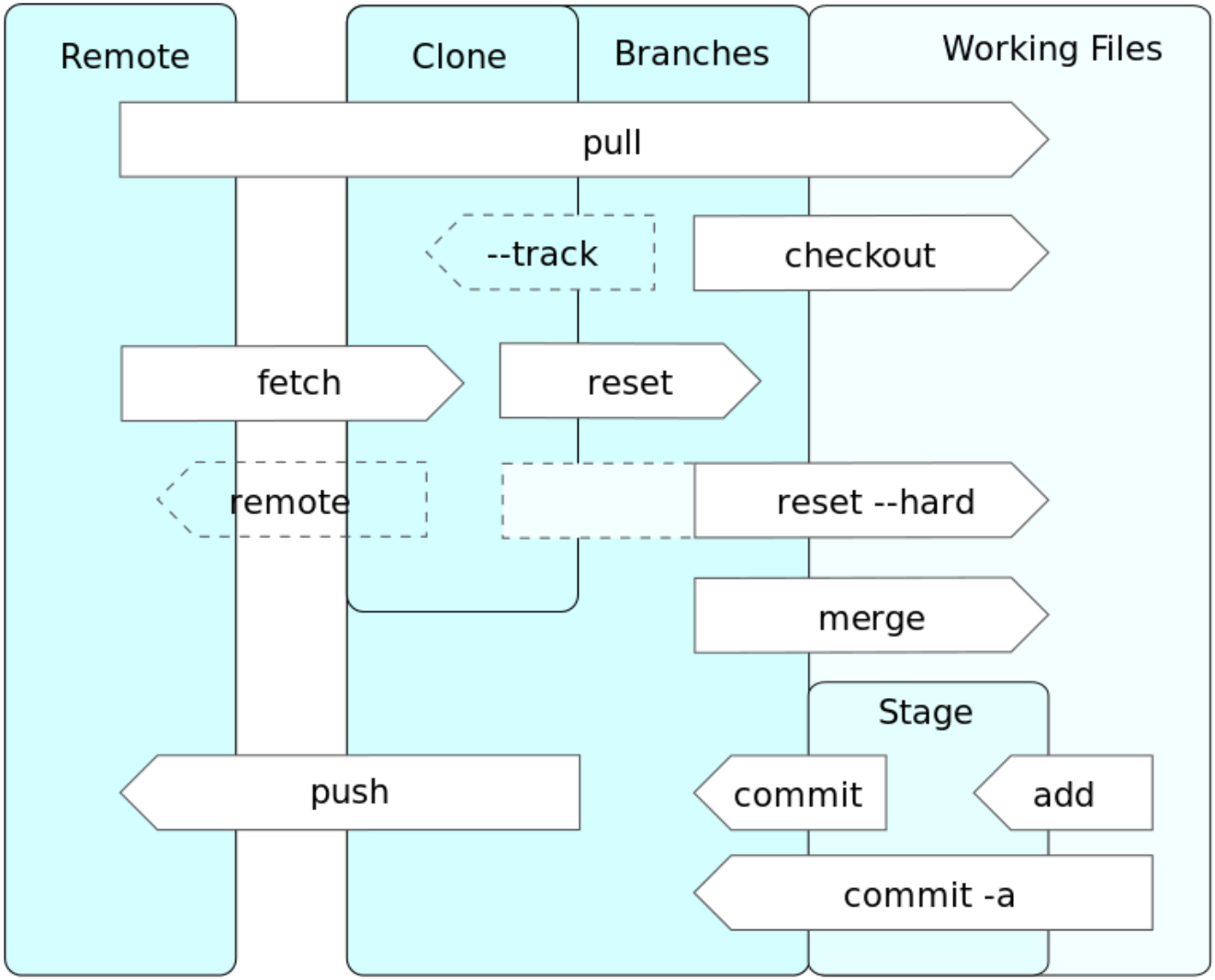
```
eiche@localhost:~/GitProjekt_test$ git log --decorate --pretty=oneline --abbrev-commit --graph --all
*   1f7cb07 (HEAD -> master) unikaten print Befehle aus beiden branches übernommen
|\
| * 09e0dd1 Hello an den Nutzer
* | d89a76b Hello die Zweite
|/
* 77e2b98 Hello die Erste
```

Visualisierung des Repositories

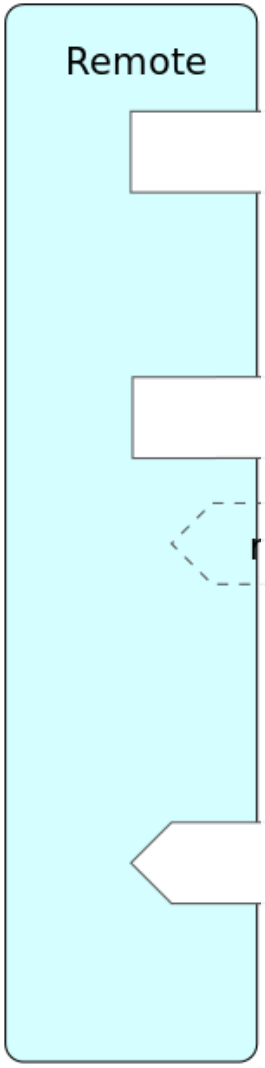
Datenfluss (im Kleinen)



Datenfluss (im Großen)



Datenfluss (im Großen)



Verteilte Versionsverwaltung:

- Haupt-Repository (*Remote*) das alle Entwickler nutzen, um ihre lokalen Repositories zu synchronisieren
- **$ git remote**
  Allgm. Konfiguration von Remotes (hinzufügen, entfernen, umbenennen, etc.)
- **$ git push**
  Commits und Referenzen in ein Remote hochladen
- **$ git pull** und **$ git fetch**
  Commits und Referenzen aus einem Remote herunterladen (git pull = git fetch + git merge)
- **$ git clone**
  Komplette Kopie herunterladen

Zweite Schritte – ein Beispiel (*Remote*)

```
eiche@was:~$ ssh -Y tic
Linux tic 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jan 29 15:46:58 2019 from 2a05:3e00:2:2020:112:3::
eiche@tic:~$ cd Dokumente/
eiche@tic:~/Dokumente$ mkdir GitTest
eiche@tic:~/Dokumente$ cd GitTest
eiche@tic:~/Dokumente/GitTest$ git init --bare --share
Leeres verteiltes Git-Repository in /home/home1/eiche/Dokumente/GitTest/ initialisiert
```

Blankes Haupt-Repository (*Remote*) erstellen

RUB

Zweite Schritte – ein Beispiel (*Remote*)

Klonen:

- Git erstellt in dem geklonten Repository für jeden *Remote-Branch* einen *Remote-Tracking-Branch.*
- Zudem wird ein *lokaler Branch* **master** erstellt, der dem *Remote-Branch* **master** entspricht.

Remote-Tracking-Branches:

- Aktuelle Zustand des Remote-Repository wird lokal gespeichert
- Remote-Tracking-Branch des *Remote-Branch* **master: origin/master**

```
eiche@dhcp-54:~$ git clone eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
Klone nach 'GitTest' ...
eiche@was.tp4.rub.de's password:
warning: Sie scheinen ein leeres Repository geklont zu haben.
eiche@dhcp-54:~/GitTest$ git remote -v
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (fetch)
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (push)
```

Remote klonen:

RUB

Zweite Schritte – ein Beispiel (*Remote*)

```
eiche@was:~$ ssh -Y tic
Linux tic 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jan 29 15:46:58 2019 from 2a05:3e00:2:2020:112:3::
eiche@tic:~$ cd Dokumente/
eiche@tic:~/Dokumente$ mkdir GitTest
eiche@tic:~/Dokumente$ cd GitTest
eiche@tic:~/Dokumente/GitTest$ git init --bare --share
Leeres verteiltes Git-Repository in /home/home1/eiche/Dokumente/GitTest/ initialisiert
```

Blankes Haupt-Repository (*Remote*) erstellen

```
eiche@dhcp-54:~$ git clone eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
Klone nach 'GitTest' ...
eiche@was.tp4.rub.de's password:
warning: Sie scheinen ein leeres Repository geklont zu haben.
eiche@dhcp-54:~/GitTest$ git remote -v
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (fetch)
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (push)
```

Remote URLs anzeigen:

RUB

Zweite Schritte – ein Beispiel (*Remote*)

ODER:

Bestehendes lokales Repository verwenden und den *Remote* setzen

```
eiche@dhcp-54:~/GitProjekt_test$ git remote add origin eiche@was.tp4.rub.de:/home/ho
me1/eiche/Dokumente/GitTest
eiche@dhcp-54:~/GitProjekt_test$ git remote -v
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (fetch)
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (push)
```

```
eiche@dhcp-54:~$ git clone eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
Klone nach 'GitTest' ...
eiche@was.tp4.rub.de's password:
warning: Sie scheinen ein leeres Repository geklont zu haben.
eiche@dhcp-54:~/GitTest$ git remote -v
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (fetch)
origin  eiche@was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest (push)
```

Remote URLs anzeigen:

Zweite Schritte – ein Beispiel (*Remote*)

```
eiche@localhost:~/GitTest$ geany hello.py &
[1] 2973
eiche@localhost:~/GitTest$ git add hello.py
[1]+  Fertig                    geany hello.py
eiche@localhost:~/GitTest$ git commit -m "Think-big-Ansatz"
[master 38db7de] Think-big-Ansatz
 1 file changed, 1 insertion(+), 2 deletions(-)
eiche@localhost:~/GitTest$ git push
eiche@was.tp4.rub.de's password:
Zähle Objekte: 3, Fertig.
Delta compression using up to 4 threads.
Komprimiere Objekte: 100% (2/2), Fertig.
Schreibe Objekte: 100% (3/3), 311 bytes | 0 bytes/s, Fertig.
Total 3 (delta 0), reused 0 (delta 0)
To was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
   90675f3..38db7de  master -> master
```

hello.py

```
1  ###Python test document
2  print("Hello world")
3  print("Hello Universe")
4
```

Änderung des lokales Repository bei Nutzer B

Zweite Schritte – ein Beispiel (*Remote*)

Remote-Name von:zu

Alternativ: **git push origin master:master**
(um den lokalen **mast**er in den *Branch* **master** im *Remote* **origin** hochzuladen)

```
eiche@localhost:~/GitTest$ g
[1] 2973
eiche@localhost:~/GitTest$ g                    geany hello.py
[1]+  Fertig
eiche@localhost:~/GitTest$ git commit -m "Think-big-Ansatz"
[master 38db7de] Think-big-Ansatz
 1 file changed, 1 insertion(+), 2 deletions(-)
eiche@localhost:~/GitTest$ git push
eiche@was.tp4.rub.de's password:
Zähle Objekte: 3, Fertig.
Delta compression using up to 4 threads.
Komprimiere Objekte: 100% (2/2), Fertig.
Schreibe Objekte: 100% (3/3), 311 bytes | 0 bytes/s, Fertig.
Total 3 (delta 0), reused 0 (delta 0)
To was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
   90675f3..38db7de  master -> master
```

Nutzer B lädt in das Remote hoch (Git entscheidet anhand der Konfigurationseinträge welche Referenzen wohin geschickt werden)

Zweite Schritte – ein Beispiel (*Remote*)

```
eiche@localhost:~/GitProjekt_test$ git fetch
eiche@was.tp4.rub.de's password:
remote: Zähle Objekte: 3, Fertig.
remote: Komprimiere Objekte: 100% (2/2), Fertig.
remote: Total 3 (delta 0), reused 0 (delta 0)
Entpacke Objekte: 100% (3/3), Fertig.
Von was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
 * branch            master     -> FETCH HEAD
```

Nutzer A lädt das Remote Repository runter (wegen fehlender Referenz zum Speichern, legt Git die Referenz in der Datei .git/FETCH_HEAD ab)

```
eiche@localhost:~/GitProjekt_test$ git merge FETCH_HEAD
Aktualisiere 90675f3..38db7de
Fast-forward
 hello.py | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Universe
```

Zweite Schritte – ein Beispiel (*Remote*)

```
eiche@localhost:~/GitProjekt_test$ git fetch
eiche@was.tp4.rub.de's password:
remote: Zähle Objekte: 3, Fertig.
remote: Komprimiere Objekte: 100% (2/2), Fertig.
remote: Total 3 (delta 0), reused 0 (delta 0)
Entpacke Objekte: 100% (3/3), Fertig.
Von was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
 * branch            master     -> FETCH HEAD
```

```
eiche@localhost:~/GitProjekt_test$ git merge FETCH_HEAD
Aktualisiere 90675f3..38db7de
Fast-forward
 hello.py | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
eiche@localhost:~/GitProjekt_test$ python hello.py
Hello world
Hello Universe
```

Nutzer A merged die Änderungen aus **FETCH_HEAD**

Zweite Schritte – ein Beispiel (*Remote*)

ODER:

```
eiche@localhost:~/GitProjekt_test$ git fetch origin
eiche@was.tp4.rub.de's password:
remote: Zähle Objekte: 3, Fertig.
remote: Komprimiere Objekte: 100% (2/2), Fertig.
remote: Total 3 (delta 0), reused 0 (delta 0)
Entpacke Objekte: 100% (3/3), Fertig.
Von was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
   3ddd00b..44514a8  master     -> origin/master
eiche@localhost:~/GitProjekt_test$ git diff master origin/master
diff --git a/hello.py b/hello.py
index e161bec..f502c2f 100644
--- a/hello.py
+++ b/hello.py
@@ -1,4 +1,3 @@
 ###Python test document
 print("Hello world")
-print("Hello Bochum")
-print("Hello user")
+print("Hello Universe")
eiche@localhost:~/GitProjekt_test$ git merge origin/master
Aktualisiere 3ddd00b..44514a8
Fast-forward
 hello.py | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

Nutzer A lädt das Remote Repository runter (dabei wurde der *Branch* **master** aus dem *Remote* verwendet, um den sog. *Remote-Tracking-Branch* **origin/master** zu aktualisieren)

RUB

Zweite Schritte – ein Beispiel (*Remote*)

ODER

```
eiche@localhost:~/GitProjekt_test$ git fetch origin
eiche@was.tp4.rub.de's password:
remote: Zähle Objekte: 3, Fertig.
remote: Komprimiere Objekte: 100% (2/2), Fertig.
remote: Total 3 (delta 0), reused 0 (delta 0)
Entpacke Objekte: 100% (3/3), Fertig.
Von was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
   3ddd00b..44514a8  master     -> origin/master
eiche@localhost:~/GitProjekt_test$ git diff master origin/master
diff --git a/hello.py b/hello.py
index e161bec..f502c2f 100644
--- a/hello.py
+++ b/hello.py
@@ -1,4 +1,3 @@
 ###Python test document
 print("Hello world")
-print("Hello Bochum")
-print("Hello user")
+print("Hello Universe")
eiche@localhost:~/GitProjekt_test$ git merge origin/master
Aktualisiere 3ddd00b..44514a8
Fast-forward
 hello.py | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

Unterschiede zwischen dem lokalen **master** *Branch* und dem *Remote-Tracking-Branch* **origin/master** anzeigen

RUB

Zweite Schritte – ein Beispiel (*Remote*)

ODER

```
eiche@localhost:~/GitProjekt_test$ git fetch origin
eiche@was.tp4.rub.de's password:
remote: Zähle Objekte: 3, Fertig.
remote: Komprimiere Objekte: 100% (2/2), Fertig.
remote: Total 3 (delta 0), reused 0 (delta 0)
Entpacke Objekte: 100% (3/3), Fertig.
Von was.tp4.rub.de:/home/home1/eiche/Dokumente/GitTest
   3ddd00b..44514a8  master     -> origin/master
eiche@localhost:~/GitProjekt_test$ git diff master origin/master
diff --git a/hello.py b/hello.py
index e161bec..f502c2f 100644
--- a/hello.py
+++ b/hello.py
@@ -1,4 +1,3 @@
 ###Python test document
 print("Hello world")
-print("Hello Bochum")
-print("Hello user")
+print("Hello Universe")
eiche@localhost:~/GitProjekt_test$ git merge origin/master
Aktualisiere 3ddd00b..44514a8
Fast-forward
 hello.py | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

Nutzer A merged die Änderungen aus **origin/master**

…und vieles mehr

- Zur besseren Visualisierung von Repositories: **gitk**
- Zahlreiche weitere Features/ Kommandos:
  - **$ git tag**

    Git-Objekte markieren, um markante Zustände hervorzuheben
  - **$ git revert**

    Änderungen eines Commit rückgängig machen (aber kein löschen von Informationen aus der Versionsgeschichte →← **$ git reset**)
  - **$ git rebase <referenz>**

    Kopiert die Commits des aktuellen Branch als neue Commits in den <referenz>-Branch (Liniearisierung des Projektverlaufs)
  - ...und vieles mehr (Haenel & Plenz (2014), https://git-scm.com/doc)

RUB

Git-hosting sites

- Stellen Infrastruktur bereit (→← eigener Git Server)
- Projekt schnell, einfach aufzusetzen, keine Server Wartung
- *Open-source code* nach außen zugänglich machen
- Liste an Git-hosting sites: https://git.wiki.kernel.org/index.php/GitHosting

Provider	Framework is open-source?	Support for other SCM	Open-source repositories	Space (GB)	Free private repositories
Beanstalk	No	SVN	No	0.1	1 projects, 1 user
bitbucket.org	No	Mercurial	Yes	Unlimited	Unlimited projects, 5 collaborators
Codebase	No	Mercurial/SVN	Public access available	0.05	1 project (unlimited repos), 2 collaborators
Helix TeamHub	No	Mercurial/SVN	No	1	Unlimited projects, 5 users
GitHub	No	SVN	Yes	Unlimited	Unlimited projects, 3 collaborators
GitLab.com	Yes	No	Yes	Unlimited	Unlimited projects, unlimited collaborators
Pikacode	No	Mercurial	Yes	1	No
ProjectLocker	No	SVN	Read-only http	0.05	1 project, 1 collaborators
repo.or.cz	Yes	No	Yes	0.4	No
RocketGit	Yes	No	Yes	Unlimited	Unlimited
SourceForge.net	Yes	Hg, SVN	Yes	Unlimited	No
Unfuddle	No	SVN	Yes	0.2	1 project, 2 collaborators
Visual Studio Team Services	No	TFVC	No	Unlimited	Unlimited, 5 users

RUB

Git-hosting sites

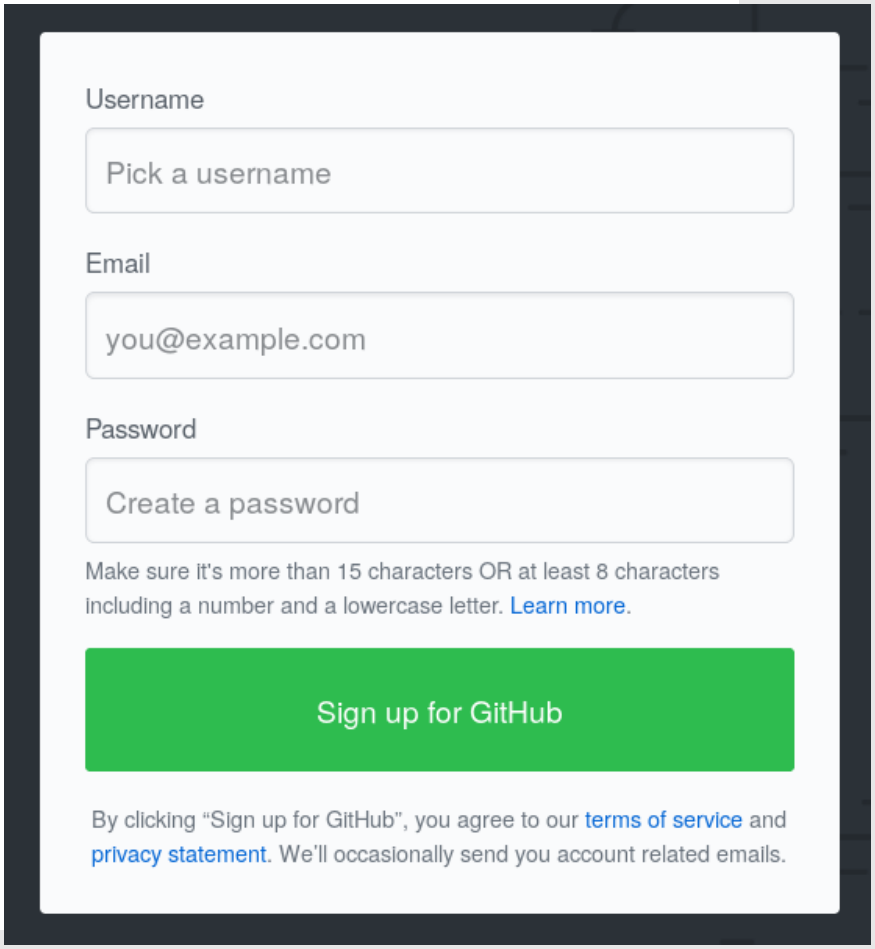
- Stellen Infrastruktur bereit (→← eigener Git Server)
- Projekt schnell, einfach aufzusetzen, keine Server Wartung
- *Open-source code* nach außen zugänglich machen
- Liste an Git-hosting sites: https://git.wiki.kernel.org/index.php/GitHosting

verbreitetste Git-Host: 28 Mio. Nutzer/ Repositories (Stand: 2018)

Provider	Framework is open-source?	Support for other SCM	Open-source repositories	Space (GB)	Free private repositories
Beanstalk	No	SVN	No	0.1	1 projects, 1 user
bitbucket.org	No	Mercurial	Yes	Unlimited	Unlimited projects, 5 collaborators
Codebase	No	Mercurial/SVN	Public access available	0.05	1 project (unlimited repos), 2 collaborators
Helix TeamHub	No	Mercurial/SVN	No	1	Unlimited projects, 5 users
GitHub	No	SVN	Yes	Unlimited	Unlimited projects, 3 collaborators
GitLab.com	Yes	No	Yes	Unlimited	Unlimited projects, unlimited collaborators
Pikacode	No	Mercurial	Yes	1	No
ProjectLocker	No	SVN	Read-only http	0.05	1 project, 1 collaborators
repo.or.cz	Yes	No	Yes	0.4	No
RocketGit	Yes	No	Yes	Unlimited	Unlimited
SourceForge.net	Yes	Hg, SVN	Yes	Unlimited	No
Unfuddle	No	SVN	Yes	0.2	1 project, 2 collaborators
Visual Studio Team Services	No	TFVC	No	Unlimited	Unlimited, 5 users

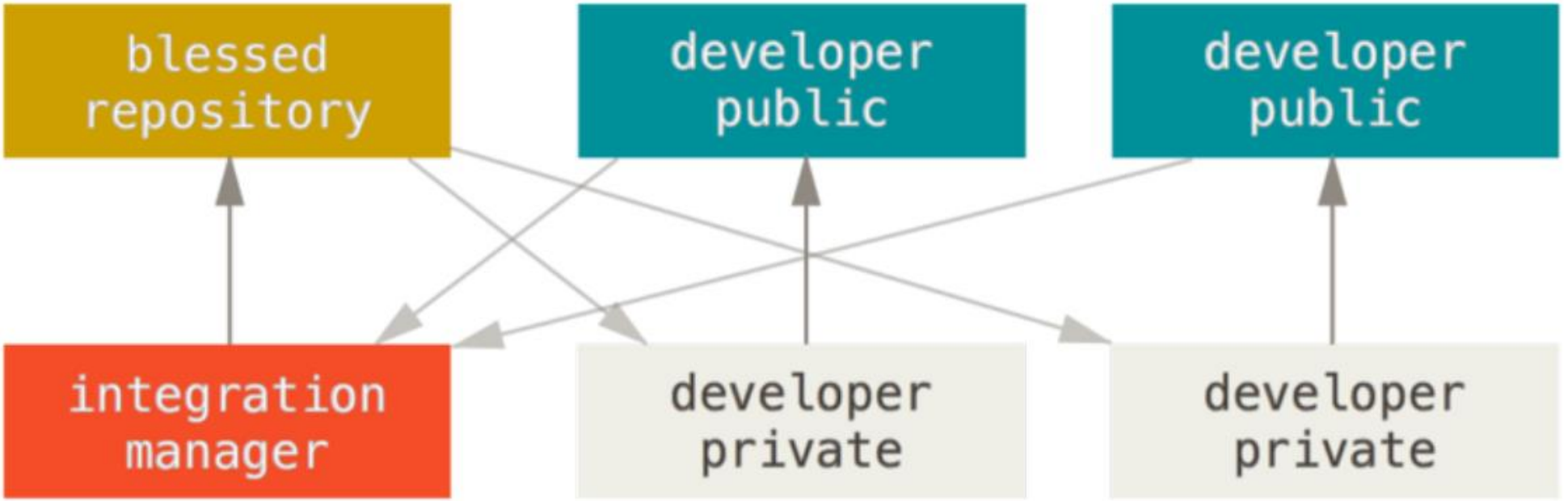
GitHub

- Git hosting, issue tracking, code review, …
- Grafische Benutzeroberfläche für diverse Anwendungen
  - Vergleich von Branches/ Repositories (anstatt: **$ git diff**)
  - *Pull Request* via Website (anstatt: **$ git request-pull**)
- Frei anmelden unter https://github.com/



GitHub – Arbeitsweise (allgemein)

1. Projekt-Besitzer stellt ein **öffentliches Repository**
2. Projekt-Teilnehmer ***klonen*** das Repository und *ändern es ab*
3. Teilnehmer ***pushen*** zur ihrer **eigenen öffentlichen Kopie**
4. Teilnehmer stellt beim Besitzer die Anfrage diese zu *pullen*
5. Besitzer *fügt das abgeänderte Repository lokal hinzu* und ***merged*** es
6. Besitzer ***pushed*** die *gemergeden* Änderungen **ins Haupt-Repository**



GitHub – Arbeitsweise (konkret)

1. Ein Projekt *forken* 

2. Einen neuen *(Themen-)Branch* vom **master** erzeugen
3. *Commits* erzeugen, die das Projekt verbessern
4. Diesen Branch auf eigenes GitHub Projekt *pushen*
5. Auf GitHub Seite *Pull-Request* stellen 

6. Änderungen diskutieren, eventuell weiter *comitten*
7. Projekt-Besitzer *merged* und schließt den *Pull-Request*

1., 5.-7.: auf GitHub

2.-4.: im lokalen Repository

RUB

GitHub – Arbeitsweise (Beispiel)

2. Einen neuen *(Themen-)Branch* vom **master** erzeugen

```
$ git clone https://github.com/tonychacon/blink ①
Cloning into 'blink'...

$ cd blink
$ git checkout -b slow-blink ②
Switched to a new branch 'slow-blink'
```

① Clone our fork of the project locally

② Create a descriptive topic branch

GitHub – Arbeitsweise (Beispiel)

3. *Commits* erzeugen, die das Projekt verbessern

```
$ sed -i '' 's/1000/3000/' blink.ino (macOS) ③
# If you're on a Linux system, do this instead:
# $ sed -i 's/1000/3000/' blink.ino ③

$ git diff --word-diff ④
diff --git a/blink.ino b/blink.ino
index 15b9911..a6cc5a5 100644
--- a/blink.ino
+++ b/blink.ino
@@ -18,7 +18,7 @@ void setup() {
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  [-delay(1000);-]{+delay(3000);+}                // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  [-delay(1000);-]{+delay(3000);+}                // wait for a second
}
```

③ Make our change to the code

④ Check that the change is good

⑤ Commit our change to the topic branch

```
$ git commit -a -m 'three seconds is better' ⑤
[slow-blink 5ca509d] three seconds is better
 1 file changed, 2 insertions(+), 2 deletions(-)
```

RUB

GitHub – Arbeitsweise (Beispiel)

4. Diesen Branch auf eigenes GitHub Projekt *pushen*

⑥ Push our new topic branch back up to our GitHub fork

```
$ git push origin slow-blink ⑥
Username for 'https://github.com': tonychacon
Password for 'https://tonychacon@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 340 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/tonychacon/blink
 * [new branch]      slow-blink -> slow-blink
```

GitHub – Arbeitsweise (Beispiel)

4. Diesen Branch auf eigenes GitHub Projekt *pushen*

⑥ Push our new topic branch back up to our GitHub fork

```
$ git push origin slow-blink ⑥
Username for 'https://github.com': tonychacon
Password for 'https://tonychacon@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 340 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/tonychacon/blink
 * [new branch]      slow-blink -> slow-blink
```

…anschließend auf GitHub Seite *Pull-Request* stellen.
(für weiter Details siehe: https://git-scm.com/doc)

Ausblick

Nächste Vorlesung

(Montag, 01.04.19, 8:30Uhr, **HGB30**):

Einführung in C++