Versionsverwaltung mit GIT

~~Michaela Meier~~
Dennis Reuling
dreuling@informatik.uni-siegen.de

Praktikum Effizientes Programmieren (PEP)
18.04.2017

Geschichte und Allgemeines

- Git ist ein Versionskontrollsystem (VCS) für Dokumente (i.d.R. Programmcode)

- Keine Weiterentwicklung von RCS,CVS,SVN

- Basiert auf neuen Konzepten
- arbeitet schneller beim Übertragen von Änderungen an vielen Dateien
- ermöglicht offline Versionskonstrolle

- Relativ junges Projekt; initiiert 2005 (Linus Torvalds)
  - Ersetzt BitKeeper (VCS zur Entwicklung des Linux-Kernels), welches nicht mehr kostenfrei zur Verfügung stand.

Teil I:
Grundlegende Konzepte

GIT Datenmodell vs. SVN Datenmodell

GIT	SVN
▪ Keine Versionsnummerierung. ▪ Stattdessen Commit-ID (Hashwert)	▪ Revisionsnummervergabe pro Projekt od. einzelner Dateien im Remote-Repository
▪ Remote- und Lokal-Repository zeigen Anzahl aller Folge-Commits direkt an	▪ Revisionsnummer erlaubt Rückschluss auf Anzahl der Folge-Commits nach
▪ Merge-"Gedächtnis": Versionsgraph ist vorhanden. So ist sofort ersichtlich, aus welchen Branches bestimmte Änderungen stammen.	▪ Kein Merge-“Gedächtnis“: Versionsgraph muss indirekt über Revisionsnummer nachvollzogen werden. Erkennung der Herkunft von Änderungen beim branch-merging schwierig.
▪ Fehler die in einem bestimmten Branch entdeckt und korrigiert wurden können in anderen Branches leicht lokalisiert werden	▪ Fehlersuche / Korrektur schwierig.
▪ Mehrere Optionen hinsichtlich Branches: ▪ merge, **rebase**, merge **(non) fast-forward**	• Nur Merge / Revert. Alle anderen Funktionen müssen manuell vollzogen werden

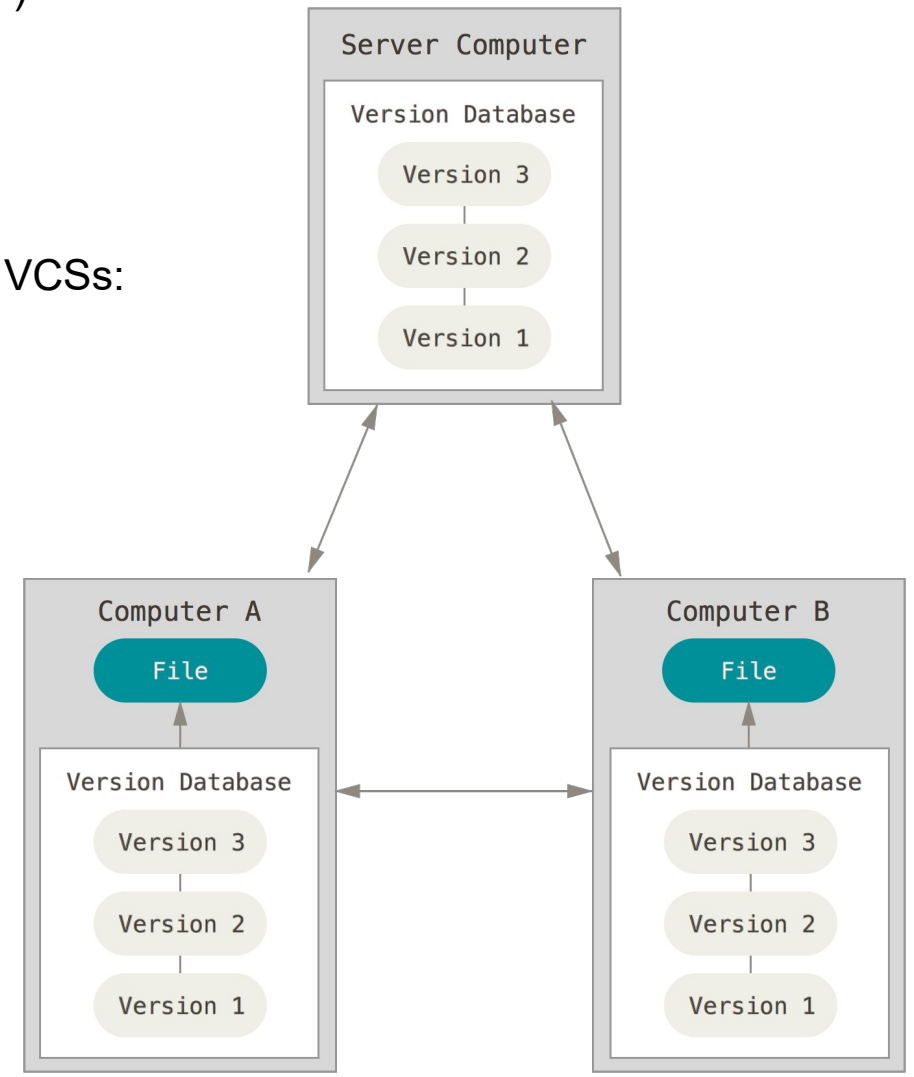
Grundlegender Unterschied zu anderen VCSs

- In GIT wird immer das komplette **Remote-Repository** in ein **Local-Repository** auf dem Computer des Entwicklers ausgecheckt (a.k.a. „**cloning**“)
  Das vollständige Abbild eines Repositories liegt lokal im **working directory.**

- Das Local-Repository besitzt alle Funktionalitäten eines VCSs:
  - comitting, branching, merging etc.

- Man arbeitet offline;
  d.h. heißt man commited in das lokale Repository

- Nach getaner Arbeit synchonisiert man alle Änderungen zwischen Remote- bzw. Local Repository



Synchronisation: Änderungen 'downloaden'

- **„fetch“**: Der fetch-Befehl ' zieht' alle Änderungen, die seit dem letzten clone/fetch/pull im Remote-Repository eingegangen sind in das lokale Verzeichnis für den remote tracking branch unter refs/remotes/<remote>/. Dabei wird das Working Directory nicht aktualisiert.

- „**pull**“ (= fetch & merge): Der pull-Befehl 'zieht' alle Änderungen, die seit dem letzten clone/fetch/pull im Remote-Repository eingegangen sind in den aktuell ausgewählten Branch im Local Repository. Dabei wird das Working Directory aktualisiert.

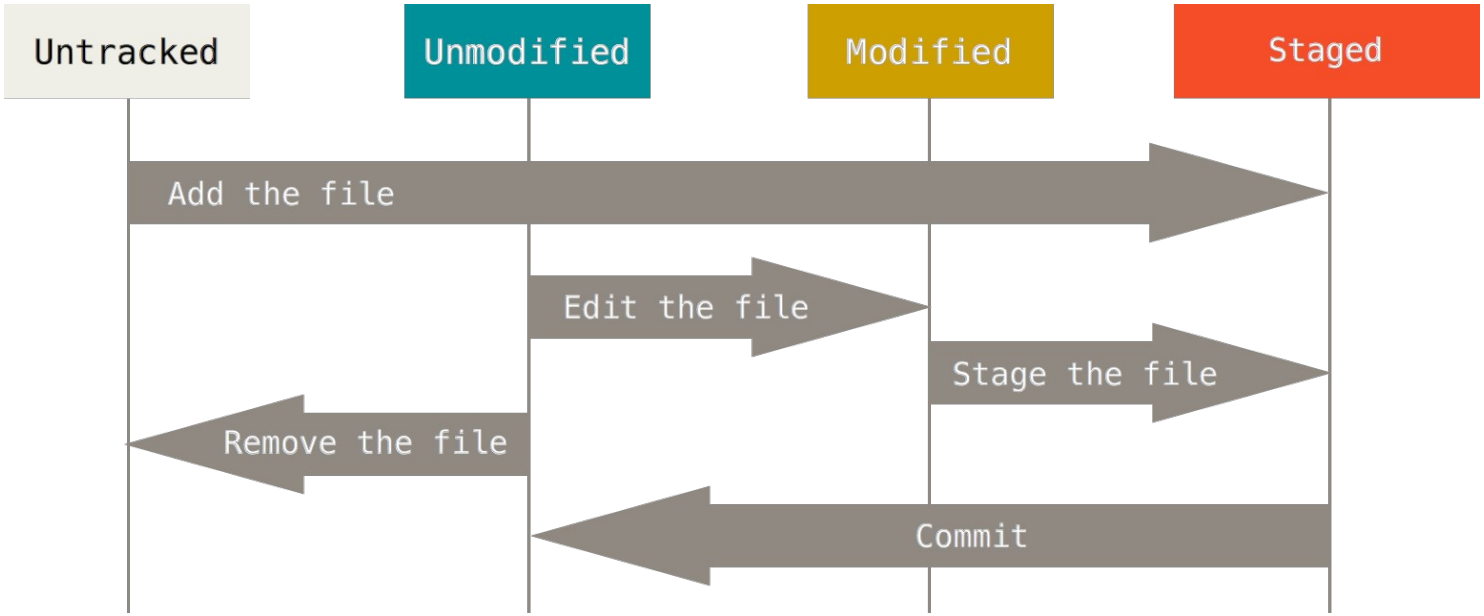
- „**checkout**“ / „**switchTo**“: ändert den aktuell ausgewählten Branch, z.B. in
  - einen remote vorhandene Branch, der zum lokalen Arbeiten ausgecheckt wird
  - einen neuen, nur lokal vorhandenen Branch

Synchronisation: Änderungen 'uploaden'

- „**staging**“: Alle Änderungen im Local-Repository, welche an das Remote-Repository gesendet werden sollen, müssen auf die Stage (engl.: Bühne) gebracht werden.

- „**commit**“: Der commit-Befehl schiebt alle ausgewählten Änderungen auf die „Stage“. Wichtig: dies geschieht offline - anders als bei anderen VCS sendet „commit“ die Änderungen nicht an den Server, sondern nur in das lokale Repository bzw. dessen Stage.

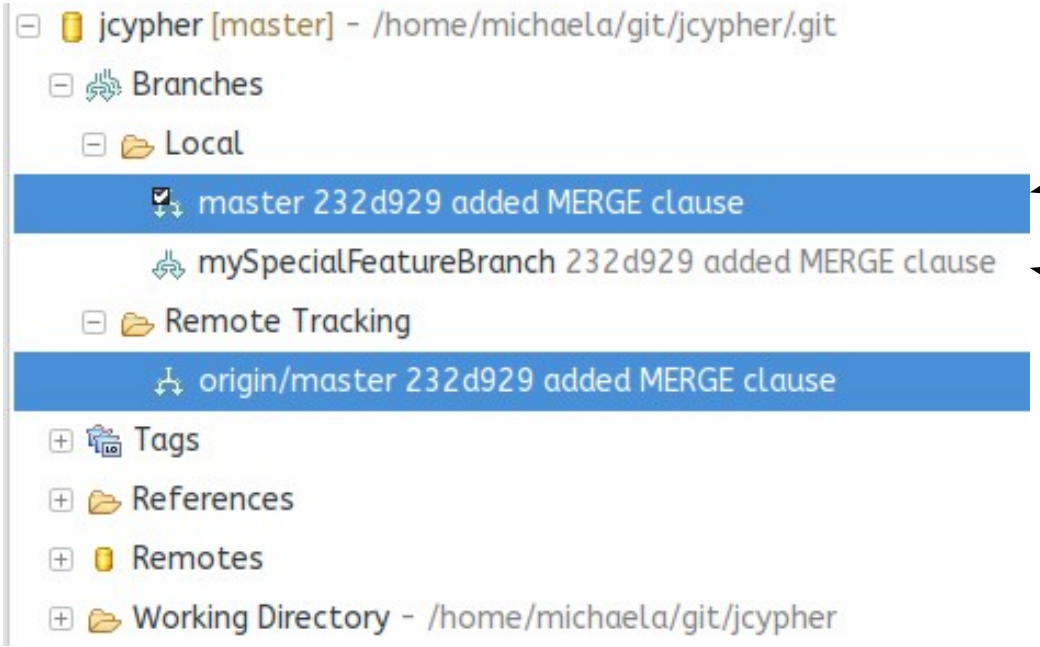
- „**push**“: Erst der „push“-Befehl sendet alle als 'staged' markierten Dateien an das Remote-Repository per push-Befehl.



Branching

Einsatz:

- Branch pro Feature/Bugfix
  - danach in den Haupt-Entwicklungszweig mischen („**merge**“)
- Branches zur Wartung älterer Versionen
- Wichtig: Branches müssen immer vor dem Arbeiten angelegt werden (nachträglich nicht möglich!)



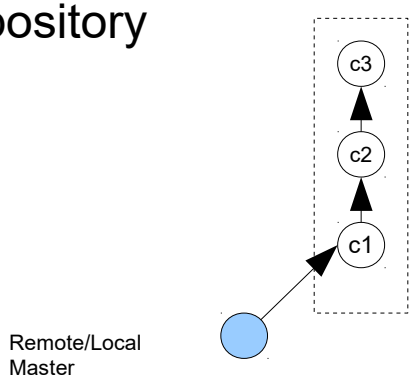
Aktuell selektierter Branch, der dem remote master entspricht

Ein 'nur' lokaler vorhandener Branch mit noch keinen committeten Änderungen

Beispiel: GIT-Repository View in egit (Eclipse)

GIT Merge Optionen

Scenario: Man möchte seine commits pushen und es gab keine Änderungen in der Zwischenzeit im Remote-Repository



lokale commits

- Weg 1: Merge mit Option **fast-foward**

  schnell und direkt

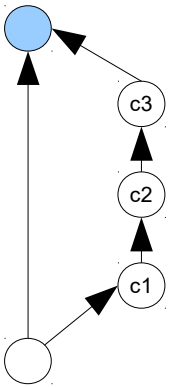
Remote/Local
Master

*branch pointer updated **fast-forward***

- Weg 2: Merge mit Option **non fast-foward**

  nicht schnell, da indirekt über separatem Branch 'Umweg'

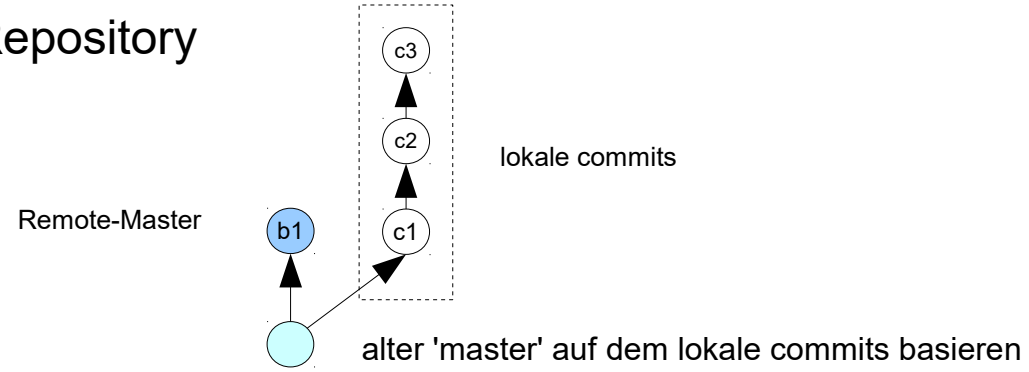
Remote/Local
Master
(merged)



Merge-node created &
branch pointer updated
(non fast-forward)

GIT Merge Optionen

Scenario: Man möchte seine commits pushen ABER es gab Änderungen in der Zwischenzeit im Remote-Repository



- Weg 1**: rebase** local branch (Abb. 1)
  + commit + push (Abb. 2)
  = Fast-forward merge

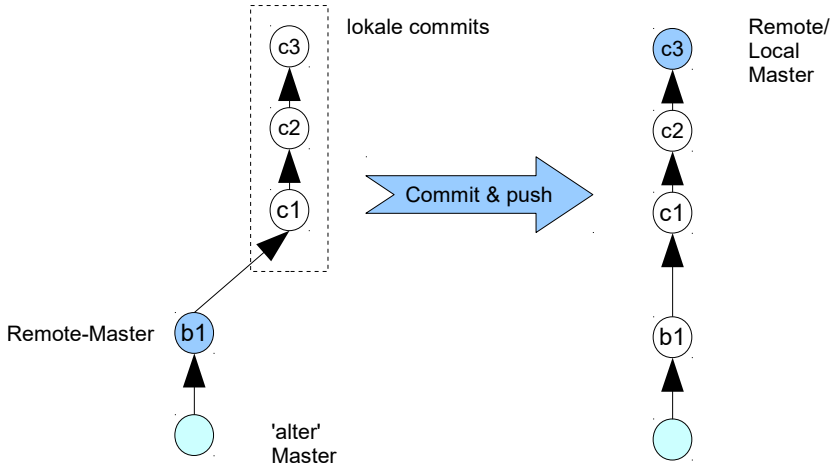


Abb. 1 (lokal: nach rebase)

Abb. 2 (lokal und remote synchron)

- Weg 2**:** (3-Way) **merge +** commit + push

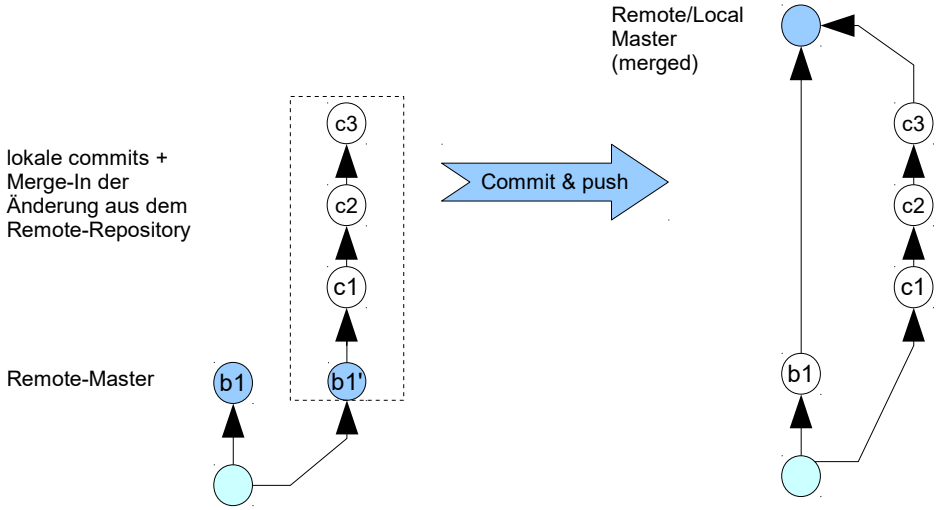


Abb. 3 (lokal und remote synchron)

Abb. 3 (lokal und remote synchron)

Zusammenführen von Branches: Merge Konflikte

- Typischer Merge-Konflikt:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

- Darstellung von Merge-Konflikten (im Prinzip wie in SVN):

```
<<<<<< < HEAD:index.html
< div  id= "footer" > contact : email.support@github.com </div>
=======
<div  id= "footer" >
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

- Konfliktauflösung: Manuell oder unterstützt durch Merge-Tool

```
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecme
Merging:
index.html
```

Teil II:
Arbeiten mit GIT

Client Software

Clients ohne GUI:

- Gängige Linux-Distributionen stellen GIT in ihrer Paketverwaltung unter dem Paketnamen git bereit
- Für andere Plattformen bietet die Git-Projekthomepage eine Download-Seite (https://git-scm.com/)
- Arbeit vorwiegend über die Kommandozeilenschnittstelle

Clients mit GUI:

- Eclipse (ab Mars) hat GIT standardmäßig integriert (als Plugin „eGIT“ http://www.eclipse.org/egit/)
- Unter Windows lässt sich außerdem Tortoise-GIT (https://tortoisegit.org/) nutzen

GIT Server:

- Wir stellen euch für dieses Praktikum einen Server bereit (s. nächste Folie)
- Ansonsten bietet sich für private Projekte (https://github.com/) an

Authentifizierung

- GIT Verbindung zu unserem Server läuft via SSH
- SSH-Verbindung benötigt SSH-Schlüssel

- Generierung eines SSH Schlüsselpaars
  - Der Key-Name soll wie folgt heißen: BENUTZER.pub
    - wobei BENUTZER dem ersten Buchstaben eures Vornamens und dem Nachnamen entsprechen sollte, also z.B. kschmidt für Karl Schmidt
  - Die Schlüssel sollen **ohne Passwort** generiert werden
  - unter Linux (vermutlich auch Mac)
    - ssh-keygen -q -t rsa -b 2048 -P "" -f BENUTZER
  - unter Windows:
    - s. http://www.howtoforge.com/ssh_key_based_logins_putty

- WICHTIG:
  - **Den SSH Public Key (nicht den Private key!) an mich schicken:** dreuling@informatik.uni-siegen.de
  - **Bis spätestens** kommenden Dienstag 25.04., 23:59, **harte Deadline!**

Remote Repository für das PEP

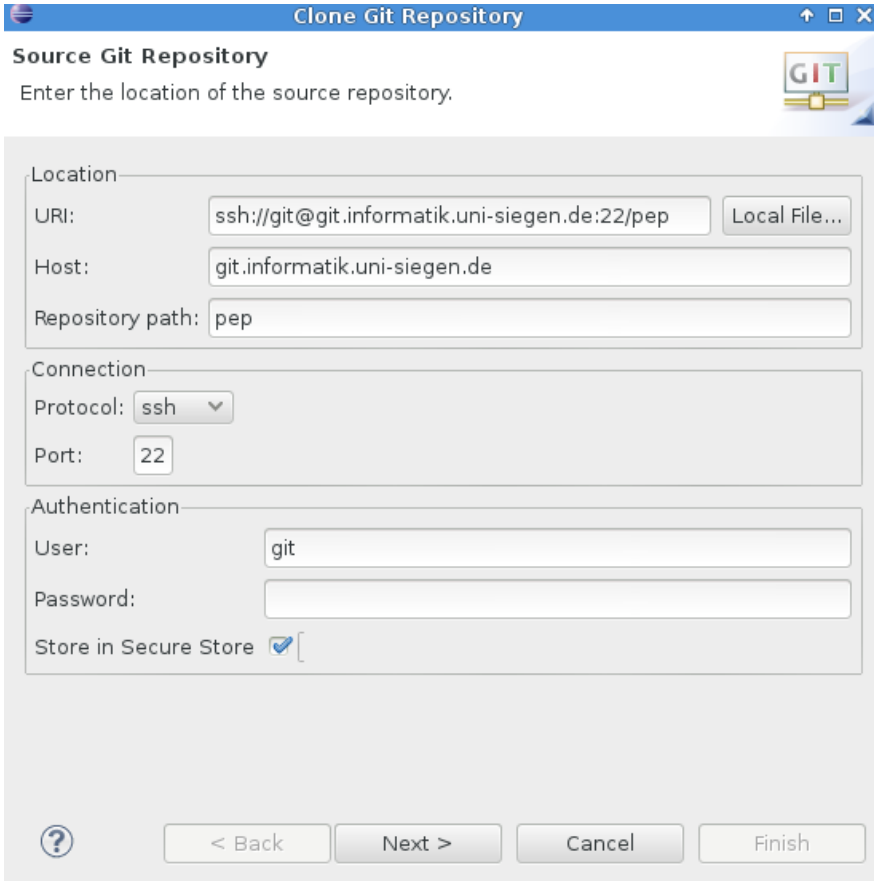
- URL: git.informatik.uni-siegen.de
- Repository: pep2017
- Protokoll: ssh
- User: git
- Password: keines (wird durch den ssh key ersetzt)

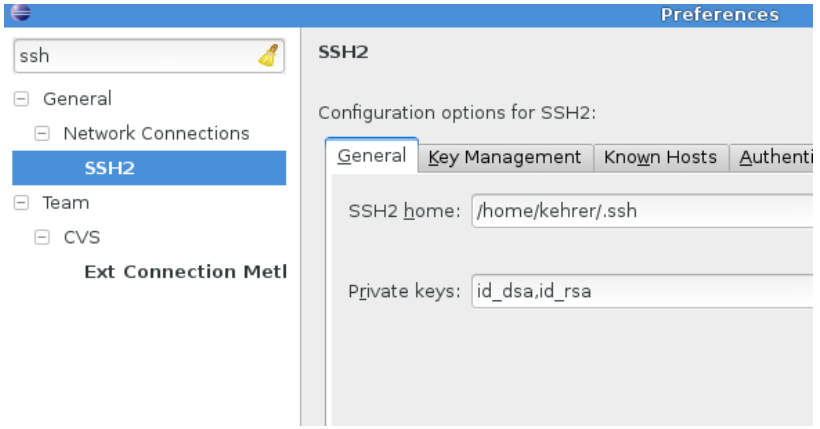
Klonen des Remote Repositories

- Spiegeln des vollständigen Remote Repositories in dein lokales Repository

```
git clone ssh://git@git.informatik.uni-siegen.de/pep2017
```

- Oder via Egit unter Eclipse:





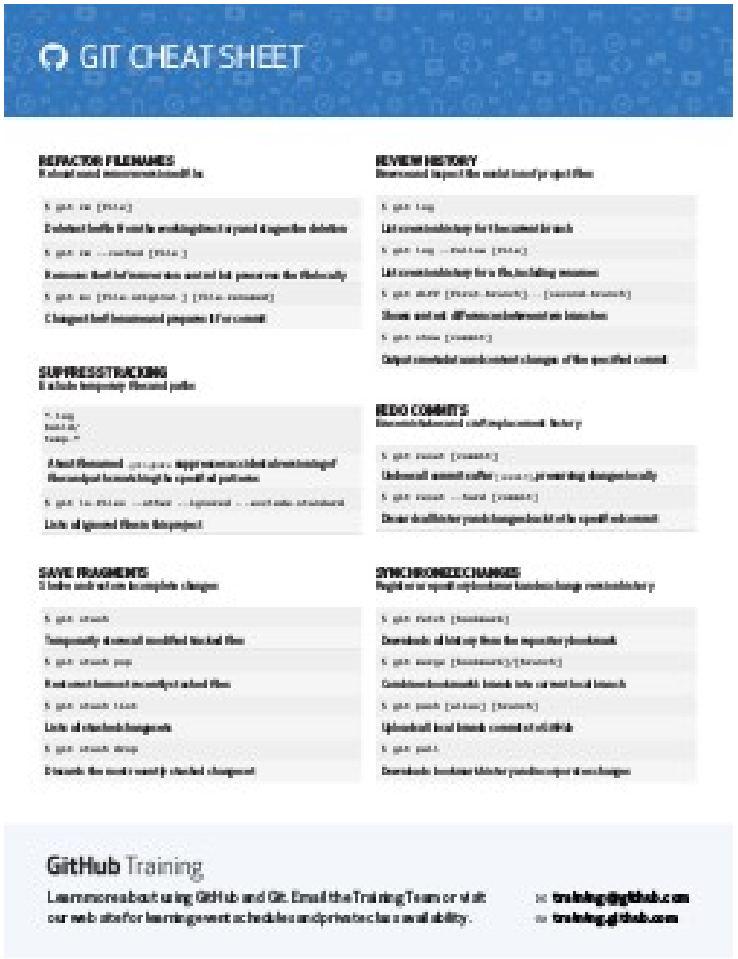
Kommando-Übersicht

```
$ git help
Die allgemein verwendeten Git-Kommandos sind:
   add          merkt Dateiinhalte zum Commit vor
   bisect       Findet über eine Binärsuche die Änderungen, die einen Fehler verursacht haben
   branch       Zeigt an, erstellt oder entfernt Branches
   checkout     Checkt Branches oder Pfade im Arbeitsverzeichnis aus
   clone        Klont ein Repository in einem neuen Verzeichnis
   commit       Trägt Änderungen in das Repository ein
   diff         Zeigt Änderungen zwischen Commits, Commit und Arbeitsverzeichnis, etc. an
   fetch        Lädt Objekte und Referenzen von einem anderen Repository herunter
   grep         Stellt Zeilen dar, die einem Muster entsprechen
   init         Erstellt ein leeres Git-Repository oder initialisiert ein bestehendes neu
   log          Zeigt Commit-Historie an
   merge        Führt zwei oder mehr Entwicklungszweige zusammen
   mv           Verschiebt oder benennt eine Datei, ein Verzeichnis, oder einen symbolischen Verweis um
   pull         Fordert Objekte von einem externen Repository an und führt sie mit einem anderen Repository
                oder einem lokalen Branch zusammen
   push         Aktualisiert Remote-Referenzen mitsamt den verbundenen Objekten
   rebase       Baut lokale Commits auf einem aktuellerem Upstream-Branch neu auf
   reset        Setzt aktuellen HEAD zu einem spezifizierten Zustand
   rm           Löscht Dateien im Arbeitsverzeichnis und von der Staging-Area
   show         Zeigt verschiedene Arten von Objekten an
   status       Zeigt den Zustand des Arbeitsverzeichnisses an
   tag          Erzeugt, listet auf, löscht oder verifiziert ein mit GPG signiertes Tag-Objekt
```

Kommando-Übersicht

- s. z.B. auch GIT Cheat Sheet von GitHub

  (https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf)



„Learning by Doing“

- **Übung** zu diesem Thema ist:

  - GIT verstehen und den Umgang damit eigenständig erlernen
  - Hilfestellung: **Tutorials** aus dem freiesMagazin Git Teil 1 -3 (s. Literatur)

  - Außerdem:
    - Kommende Themen werden z.T. weitere **Übungen** anbieten, in denen Programmcode kollaborativ in GIT erstellt/bearbeitet werden soll.

Literatur

- Git vs SVN - Eine vergleichende Einführung
  http://de.slideshare.net/mariomueller/git-vs-svn-eine-vergleichende-einfhrung

- GIT Cheat Sheet von GitHub
  https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf

- Scott Chacon and Ben Straub: „Pro Git book“
  https://git-scm.com/book/en/v2

- freiesMagazin 12/2014: GIT-Tutorium Teil 1
  http://www.freiesmagazin.de/mobil/freiesMagazin-2014-12-bilder.html#fm_14_12_git_teil1

- freiesMagazin 01/2015: GIT-Tutorium Teil 2
  http://www.freiesmagazin.de/mobil/freiesMagazin-2015-01-bilder.html#fm_15_01_git_teil2

- freiesMagazin 02/2015: GIT-Tutorium Teil 3
  http://www.freiesmagazin.de/mobil/freiesMagazin-2015-02-bilder.html#fm_15_02_git_teil3

- Webseite 04/2016
  https://www.atlassian.com/git/tutorials/using-branches/git-merge

- EGit Tutorial:
  http://eclipsesource.com/blogs/tutorials/egit-tutorial/