LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

GIT Tutorial

Martin Ritter
LMU Munich

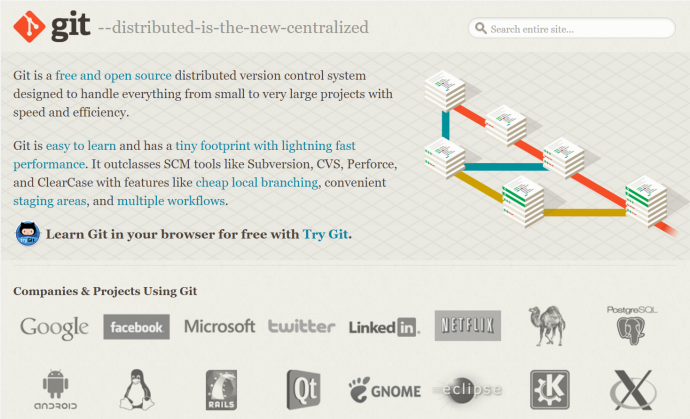
Bachelor Einführung, April 2018

TUM LMU Excellence Cluster Universe

Bundesministerium für Bildung und Forschung

Git

Verteiltes System zur Versionskontrolle seit 2005

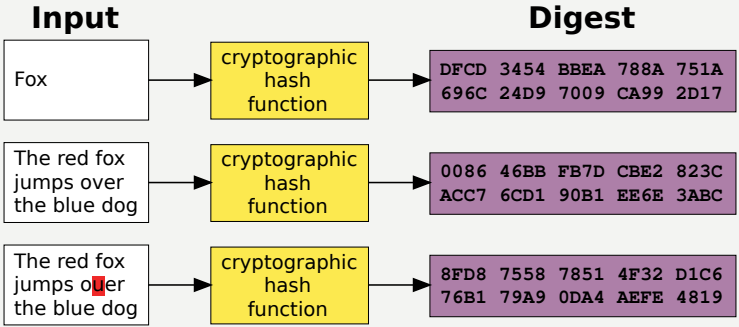


- protokolliert Änderungen an Dateien
- benötigt keinen Zentralen Server
- jeder Benutzer hat die komplette Versionsgeschichte (“clone”)
- synchronisation mit *push* und *pull*

https://git-scm.org

Kryptographische Hashfunktion

Erzeuge eine Prüfsumme mit fester länge aus beliebigen Eingaben



- kleine Änderungen an der Eingabe ⇨ große Änderungen am hash.
- **Einwegfunktion:** praktisch unmöglich den Ursprungswert aus dem hash zu bestimmen
- **Kollisionsresistenz:** praktisch unmöglich einen anderen Ursprungswert mit dem gleichen hash zu ermitteln

Die langweilige Theorie

Fangen wir mit der Theorie an:

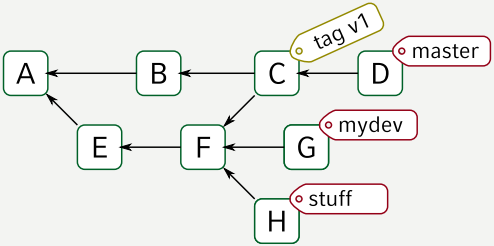
- Git verfolgt wie sich Dateien verändern
- Ihr müsst git sagen was, wann und warum
- Ein solcher Zustand heißt **Commit** und enthält
  - das Datum
  - den Namen und Inhalt aller Dateien
  - den Ersteller des commit
  - eine Nachricht warum der Commit erstellt wurde
  - Verweise auf ein oder mehrere vorangehende commits.

Commit ID

- SHA1 hash des ganzen commit (Dateien + Information)
- jedwede Änderung am commit würde die ID ändern.
- üblicherweise abgekürzt zu 6-8 Zeichen.

Git Commit Baum

Alle git commits bilden eine Baumstruktur (Hash-Baum, “merkle tree”)



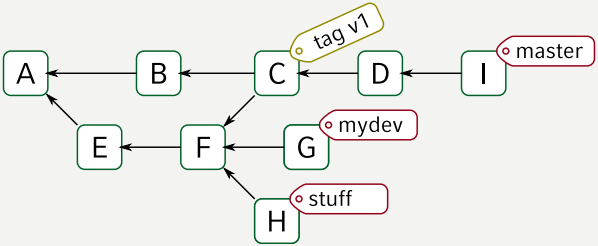
Referenzen

- Ein **Branch** zeigt auf einen beliebigen Commit und wird versetzt wenn ein neuer Commit zum Branch hinzugefügt wird.
- Ein **Tag** einen beliebigen Commit, kann eine Nachricht enthalten.

- `git clone` erzeugt eine komplette Kopie des Baumes und einen Verweis auf den Ursprung
- `git push`/`git pull` synchronisiert verschiedene Bäume so effizient wie möglich

Git Commit Baum

Alle git commits bilden eine Baumstruktur (Hash-Baum, “merkle tree”)



Referenzen

- Ein **Branch** zeigt auf einen beliebigen Commit und wird versetzt wenn ein neuer Commit zum Branch hinzugefügt wird.
- Ein **Tag** einen beliebigen Commit, kann eine Nachricht enthalten.

- `git clone` erzeugt eine komplette Kopie des Baumes und einen Verweis auf den Ursprung
- `git push`/`git pull` synchronisiert verschiedene Bäume so effizient wie möglich

First Git Hands on

Go To

https://try.github.io

and Follow the Instructions™

Commits Erzeugen

Git hat eine separate "staging area"

- alle für einen Commit vorgemerkten Inhalte
- `git add` ist für jeden Commit notwendig.
- Alternativ: `git commit -a` für "Alle Änderungen"
- `git status` zeigt alle den Status aller Dateien

Pro Tip: `git add -p .`

- `git add -u .` für Dateien im aktuellen Verzeichnis die Git schon kennt.
- `git add -p .` frägt für jede einzelne Änderung

- `git commit` öffnet einen Texteditor um die Nachricht zu verfassen
- Editor konfigurieren: `git config --global core.editor "vim"` oder `export EDITOR=vim`

➡ `git gui` für eine Graphische Oberfläche zum Commit erzeugen

Arbeiten mit Branches

- `git branch` zeigt eine Liste aller Branches
- `git checkout -b branchname` erzeugt einen neuen Branch und wechselt dorthin
- `git checkout branchname` wechselt zum Branch
- Branches in anderen Repositories mit Prefix: "origin/master" ist Branch "master" in repository "origin"

In einen entfernten Branch wechseln

Wenn ein branch mit dem gleichen Namen auf dem server existiert: `git checkout branchname` wechselt in eine lokale kopie dieses Branches

Branches können einen "Upstream" haben:

- `git push` schiebt Änderungen des aktuellen Branch in den upstream Branch
- automatich gesetzt wenn der Branch von einem entfernten Branch erstellt wurde
- wenn der Branch lokal erzeugt wurde: `git push --set-upstream origin branchname`

Git Services

GitHub, GitLab, BitBucket

- Webdienste für Git
- Einfach erreichbar
- Repositories/Branches verwalten.
- Pull/Merge Requests

➡ `https://gitlab.lrz.de` mit LRZ account.

Zugriffsmethoden

- der einfachste Weg: SSH schlüssel
- alternativ mit Benutzername/Passwort

SSH Configuration

SSH Configuration

- ~/.ssh/config
- Kurze Namen:
  ssh kekcc
- Schlüsselwahl
- Zwischenverbindungen
- Geteilte Verbindungen
- ...

➡ Sehr empfehlenswert

```
# use control hub to use only one ssh channel for all connections.
# faster connection and password only needed for first connection
ControlMaster auto
ControlPath ~/.ssh/%r@%h:%p.control
ControlPersist 10m
# make all locally present ssh keys available at the remote site
ForwardAgent yes
# try to keep the connection alive, this avoids connection timeouts
ServerAliveInterval 60

Host kekcc
  User ritter
  Hostname login.cc.kek.jp
  IdentityFile ~/.ssh/config/private-key-filename
  Compression yes
  # Don't connect directly but rather via the login server
  ProxyCommand ssh sshcc1.kek.jp -W %h:%p
  # since OpenSSH 7.3: ProxyJump sshcc1.kek.jp

Host sshcc1.kek.jp
  User ritter
```

Übung: Branches in der Kommandozeile

Auschecken

- erstellt clone von `https://gitlab.lrz.de/ritter/git_tutorial_18.04`
- optional per ssh

- welche Änderungen wurden als letztes vorgenommen?
- zeige eine Liste aller Branches auf dem Server
- zeige die Änderungen zwischen den branches `master` und `branch_merge`
- merge von `branch_merge` nach `master`
- merge von `branch_conflict` nach 'master

Übung: Branches im Webfronted

Alle vorherigen Aufgaben lassen sich auch komfortabel über das Webfrontend erledigen

Auschecken

- einloggen in `https://gitlab.lrz.de`
- erstellt fork von `https://gitlab.lrz.de/ritter/git_tutorial_18.04`

- welche Änderungen wurden als letztes vorgenommen?
- zeige eine Liste aller Branches auf dem Server
- zeige die Änderungen zwischen den branches `master` und `branch_merge`

Pull/Merge Requests

Um Änderungen kollaborativ einzufplegen hat sich der “Pull/Merge Request” Arbeitsablauf eingebürgert:

- vorbereiten aller Änderungen in einem Branch
- anfrage eines “Merge” dieses Branch über webfrontend
- zuweisung von Reviewern, integrations checks, Diskussion
- anpassen der Änderungen
- einplfegen nach Konsens

Erstellen eines merge request

- erstellen eines merge request von `branch_conflict` auf den geforkten `master`
- lösen der Konflikte
- kommentar erstellen
- merge durchführen

Große Dateien in git

Git synchronisiert immer den kompletten Baum:

- große Dateien werden immer heruntergeladen
- **auch nachdem sie gelöscht wurden**

⇨ Keine großen Dateien direkt in git

Git LFS

- `git lfs` ist eine Erweiterung die große Dateien extern speichert
- muss im repository installiert & evtl. im webfrontend aktiviert werden
- wenn unbedingt große Dateien im git repository sein müssen

Thank you
for your attention