Versionskontrolle: Git (Einstieg)

BBZW, Modul 426

Patrick Bucher

29.02.2024

Geschichte

Die Entwickler des Linux-Kernels tauschten zunächst Patches (`diff`) als Tarballs (`*.tar.gz`) untereinander aus.

Später stiegen sie auf das kommerzielle VCS *BitKeeper* um. Ab 2005 wurde dies jedoch den Kernel-Entwicklern nicht mehr kostenlos zur Verfügung gestellt.

Linus Torvalds entschied sich, seine eigene Versionskontrolle **Git** (engl. *Schwachkopf*) zu entwickeln.

Merkmale

Git wurde für die Bedürfnisse der Linux-Kernel-Entwicklung mit folgenden Zielen entwickelt:

- Geschwindigkeit
- einfaches Design
- Unterstützung für nicht-lineare Entwicklung (Branching)
- vollständige Verteilung
- Effizienz im Umgang mit grossen Projekten

Fazit: *Ziel erreicht*

Git und traditionelle Versionskontrollsysteme

Traditionelle Versionskontrollsysteme

- Die meisten VCS arbeiten *dateiorientiert*.
- Das VCS speichert Änderungen zwischen einzelnen Dateien ab.
- Die Versionsgeschichte besteht aus einer *Reihe von Änderungen*.
    - Analogie: Transaktionen auf einem Bankkonto, das bei 0 startet.

Git

- Git arbeitet mit Zwischenständen (*Snapshots*).
- Git speichert bei Änderungen einen neuen Zwischenstand ab.
- Die Versionsgeschichte besteht aus einer *Reihe von Zuständen*.
    - Analogie: Dateisystem, das zu bestimmten Zeiten gesichert wird.

Traditionelle Versionskontrollsysteme: Diffs

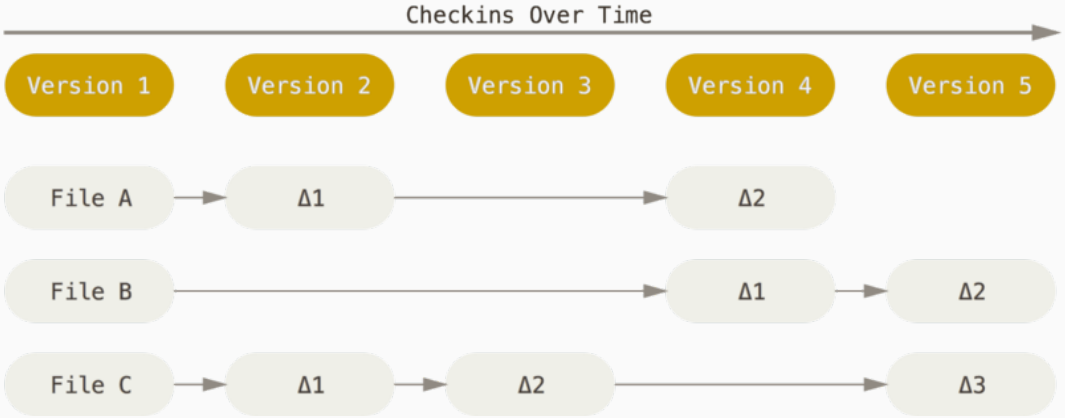

Abbildung 1: Versionsgeschichte: eine Reihe von Änderungen (https://git-scm.com/book/de/v2/Erste-Schritte-Was-ist-Git%3F)

Git: Snapshots

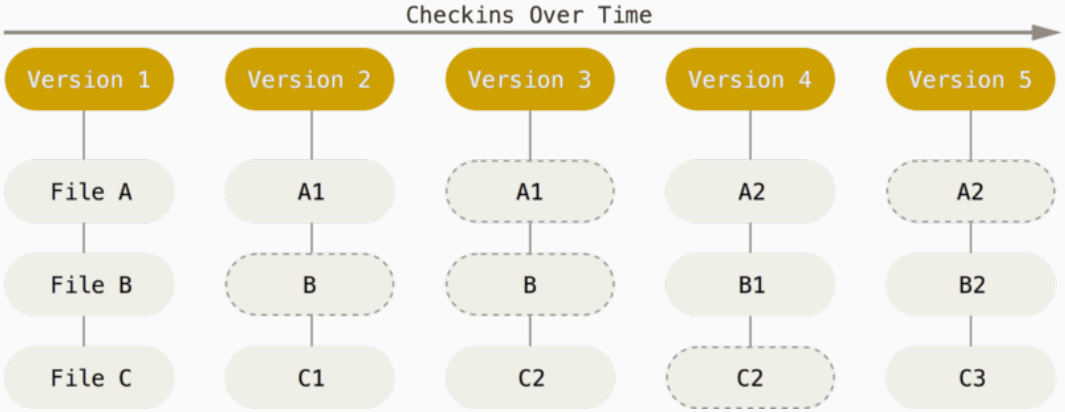

Abbildung 2: Versionsgeschichte: eine Reihe von Zuständen (https://git-scm.com/book/de/v2/Erste-Schritte-Was-ist-Git%3F)

Git: Weitere Merkmale

Ein Verzeichnis, das von Git verwaltet wird, nennt man ein **Repository** (eine Art Datenbank).

Git berechnet für jeden Zustand über alle Dateien hinweg eine **Prüfsumme** (SHA1: 40 hexedezimale Stellen): z.B. `353edf7e8d13b6535b26bcb150c47fbc6e50c0c4`

- Dadurch können Änderungen und korrupte Daten erkannt werden.
- Zustände werden eindeutig über diesen Hash referenziert.

Git funktioniert **additiv**, d.h. die meisten Änderungsoperationen fügen Daten hinzu.

- Das Repository wird dadurch tendenziell immer grösser.
- Dafür können keine festgeschriebenen Änderungen verlorengehen.

Git: Bereiche

Git unterscheidet zwischen drei Bereichen, die Zustände repräsentieren:

1. **Arbeitsverzeichnis**: unverändert oder veränderte Dateien
2. **Staging-Bereich**: vorgemerkte Änderungen
3. **`.git`-Verzeichnis**: festgeschriebene Änderungen
   - enthält Meta- und Objektdaten

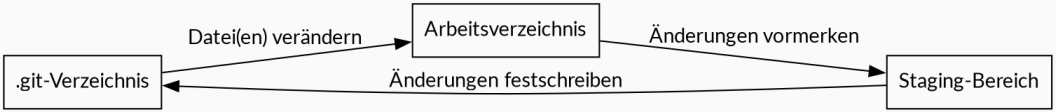

Abbildung 3: Bereiche und Zustandsübergänge

Git: Clients

- Es gibt einen offiziellen Git-Client für die Kommandozeile: `git`.
  - Dieser unterstützt *alle* Git-Befehle.
- Dazu gibt es viele GUI-basierende Git-Clients.
  - Diese unterstützen nur verschiedene Untermengen des Funktionsumfangs.
- Wer den `git`-Befehl kennt, kommt auch mit einem GUI-Client zurecht – umgekehrt gilt das nicht.

Darum wird hier nur `git` auf der Kommandozeile erläutert!

Git: Installation

Ubuntu:

```
# apt install git
```

Windows: Download

```
klick, klick, klick
```

macOS:

```
$ brew install git
```

Git: Konfigurationsdateien

Git kann auf drei Ebenen konfiguriert werden:

Bereich	Befehl	Konfigurationsdatei
systemweit	`git config --system`	`/etc/gitconfig`
pro Benutzer	`git config --global`	`~/.gitconfig` bzw. `~/.config/git/config`
Repository	`git config`	`.git/config`

Unter Windows liegt die systemweite Konfigurationsdatei im Installationsverzeichnis.

Konfigurationen können auf einer tieferen Stufe überschrieben werden.

Git: initiale Konfiguration

Jede Änderung wird *unwiderruflich* mit Name und E-Mail-Adresse des Benutzers festgeschrieben. Name und E-Mail-Adresse sollten darum gleich zu Beginn gesetzt werden:

```
$ git config --global user.name 'Vorname Nachname'
$ git config --global user.email 'vorname_nachname@sluz.ch'
```

Der Texteditor (für mehrzeilige Änderungskommentare) lässt sich auch konfigurieren (z.B. `vim`, `emacs` oder `nano`):

```
$ git config --global core.editor nano
```

Git: Konfiguration einsehen

Die für den aktuellen Kontext geltende Konfiguration abfragen:

```
$ git config --list
user.email=vorname_nachname@sluz.ch
user.name=Vorname Nachname
```

Eine einzelne Einstellung abfragen:

```
$ git config user.email
vorname_nachname@sluz.ch
```

Aufgabe: Installation & Konfiguration (ca. 15 Minuten, Einzelarbeit)

Installieren Sie auf Ihrem BYOD die Git Bash für Windows:

- Setup: mit Standardeinstellungen
- Alternative (ohne Admin-Rechte) Portable: entpacken

Konfigurieren Sie Git folgendermassen *global*:

- Name (`user.name`): `"[Vorname] [Nachname]"` (ohne `[` und `]`)
- E-Mail (`user.email`): `[vorname]_[nachname]@sluz.ch` (ohne `[` und `]`)
- Texteditor (`core.editor`): nach Ihrer Wahl und Kenntnis (`notepad`, `nano`, `vim`)

Geben Sie einen Screenshot von `git config --list` in Teams ab!

- **Zusatzaufgabe** (auf Firmenlaptop): Git with Multiple E-Mail Addresses
- **Zusatzaufgabe** (falls Sie fertig sind): Stöbern Sie in *Pro Git* (EN, DE)

Quellen und Links

- Tech Talk: Linus Torvalds on git
- Pro Git (en)
- Pro Git (de)
- Git Reference (en)