mitp

Sujeevan
Vijayakumaran

3. Auflage

Versionsverwaltung mit Git

Praxiseinstieg

Mit praktischer Referenzkarte

Inhaltsverzeichnis

Sujeevan Vijayakumaran

Versionsverwaltung mit Git

Praxiseinstieg

Impressum

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-7475-0306-5
3. Auflage 2021

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Coverbild: Verne Ho @verneho / unsplash.com,
Coverbild gestaltet von Bernhard Hanakam
electronic **pub**lication: III-satz, Husby, www.drei-satz.de

Dieses Ebook verwendet das ePub-Format und ist optimiert für die Nutzung mit dem iBooks-reader auf dem iPad von Apple. Bei der Verwendung anderer Reader kann es zu Darstellungsproblemen kommen.

Cheat Sheet

Repositorys anlegen

git init [<pfad>]

Erzeugt ein neues leeres Repository im aktuellen Arbeitsverzeichnis. Bei optionaler Angabe des Zielpfades wird ein neues Verzeichnis mit dem Repository darin angelegt.

git clone <repository> [<pfad>]

Klont das als weiteren Parameter angegebene Repository in ein neues Verzeichnis im aktuellen Verzeichnis. Zusätzlich wird es als Remote-Repository `origin` konfiguriert. Bei der Angabe eines weiteren Verzeichnisses wird das Repository unter einem anderen Verzeichnisnamen geklont.

Erweiterung und Bearbeitung der Historie

Arbeiten im Staging-Bereich

git add <pfad>

Mit diesem Befehl können Änderungen von Dateien zum Staging-Bereich hinzugefügt werden, die anschließend in einem Commit gespeichert werden können.

git reset

Setzt den aktuellen HEAD auf einen bestimmten Stand zurück. Anwendbar sowohl bei Änderungen, die noch nicht in einem Commit sind, als auch bei Änderungen in diversen Commits.

git restore

Setzt Dateien im Arbeitsverzeichnis des Repositorys zurück. Mit `--source` kann optional die Quelle angegeben werden. Mit `--staged` kann eine Änderung aus dem Staging-Bereich wieder herausgenommen werden.

Arbeiten mit Commits und Branches

git commit

Erzeugt einen Commit aus den Änderungen aus dem Staging-Bereich. Ohne zusätzlichen Parameter öffnet sich der konfigurierte Editor. Alternativ kann die Commit-Message mit `-m` direkt beim Aufruf gesetzt werden.

git branch [<name>]

Listet alle lokalen Branches auf und erzeugt einen neuen Branch mit dem angegebenen Namen. Dabei wird nicht auf den neuen Branch gewechselt.

git checkout [<branch>]

Wechselt auf den angegebenen Branch. Mit `-b` kann optional ein neuer Branch erstellt werden, auf den direkt gewechselt

wird.

git switch [<branch>]

Wechselt ebenfalls auf den angegebenen Branch und ist ab Git Version 2.23 verfügbar. Mit `-c` kann optional ein neuer Branch erstellt werden, auf den direkt gewechselt wird.

git diff

Zeigt die Unterschiede zwischen Commits, Commits und Arbeitsverzeichnisse, oder Commits und Staging-Bereich an.

git merge <branch>

Führt mindestens zwei Branches zusammen.

git rebase

Übernimmt Commits von anderen Branches und fügt diese auf dem aktuellen Branch an.

Verteilte Repositorys

git fetch

Lädt alle Objekte des konfigurierten Remote-Repositorys herunter und macht sie lokal zugänglich. Häufig werden mit dem Parameter alle Daten von allen konfigurierten Remote-Repositorys heruntergeladen.

git pull

Lädt die Änderungen vom konfigurierten Branch vom Remote-Repository **und** führt ein Merge oder Rebase durch.

git push

Schiebt die lokalen Commits des Branches auf den Server.

git remote

Ermöglicht die Konfiguration von Remote-Repositorys.

Status im Repository

git status

Zeigt den Status im Repository an und gibt sehr hilfreiche Informationen aus, welche folgenden Befehle nun ausgeführt werden können.

git log

Zeigt die Historie des Repositorys an.

Multi-Repositorys verwalten

git subtree

Ermöglicht die Einbindung von anderen Git-Repositorys in das aktuelle Repository, in dem das Subtree in das Repository gemerget wird.

git submodule

Ermöglicht ebenfalls die Einbindung von anderen Git-Repositorys in das aktuelle Repository. Die Submodule-Repositorys werden über ein Commit referenziert und eingebunden.

Sonstiges

git blame <pfad>

Zeigt für die angegebene Datei Zeile für Zeile an, wer die Zeile zuletzt modifiziert hat, inklusive Commit-ID und Zeitpunkt.

git stash

Schiebt Änderungen aus dem aktuellen Arbeitsverzeichnis in einen Zwischenspeicher und kann diese Änderungen auch erneut wieder herausholen.

git config

Ermöglicht es, die Konfiguration anzupassen.

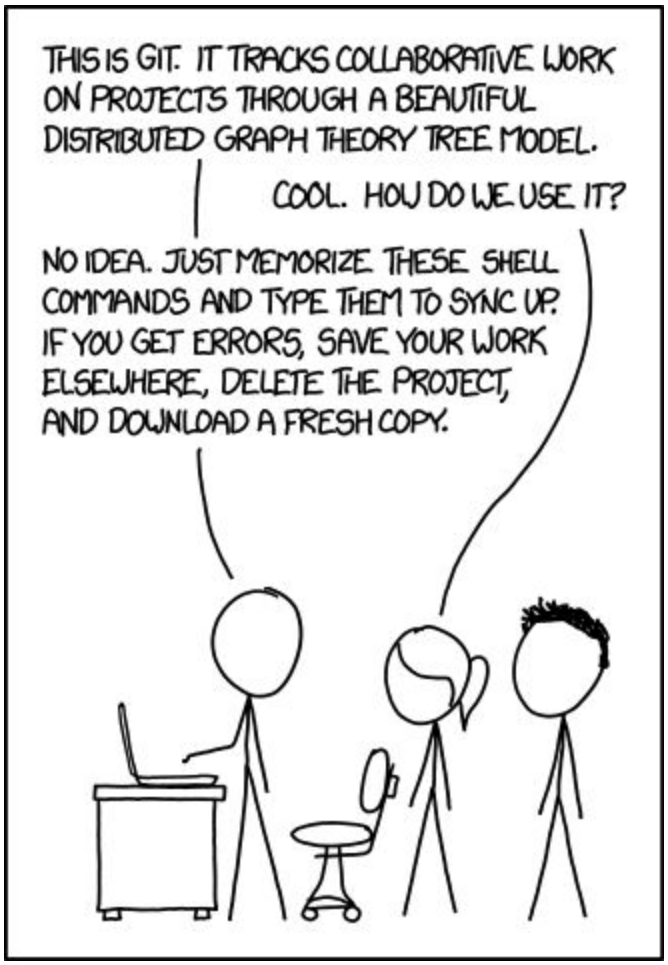
git reflog

Zeigt an, welche Tätigkeiten lokal im Repository ausgeführt wurden. Ist sehr nützlich, wenn Commits aus Versehen gelöscht wurden.

git clean

Räumt das lokale Repository auf, etwa um ignorierte oder nicht versionierte temporäre Dateien gesammelt zu entfernen.

Einleitung



»Das ist Git. Es bietet einen Überblick über die kollaborative Arbeit in Projekten durch die Nutzung eines wunderschönen Graphen-Theorie-Modells.«

Sie: »Cool. Aber wir nutzt man es?«

Er: »Keine Ahnung. Merke dir einfach all diese Befehle und tippe sie ein. Wenn du auf Fehler stößt, dann sichere deine Arbeit woanders, lösche das Projekt und lade eine frische Kopie herunter.«

»If that doesn’t fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of ›It’s really pretty simple, just think of branches as...‹ and eventually you’ll learn the commands that will fix everything.«

»Und wenn das auch nicht hilft, dann enthält git.txt die Telefonnummer von einem Freund, der sich mit Git auskennt. Warte einfach ein paar Minuten ab à la ›Es ist wirklich gar nicht so schwer, stell dir nur die Branches vor als ...‹, und schließlich lernst du die Befehle, die jedes Problem fixen.«[1]

Versionskontrolle ist ein wichtiges Thema für Software-Entwickler. Jeder, der ohne jegliche Versionskontrollprogramme arbeitet, ist vermutlich schon einmal an den Punkt gestoßen, wo man sich ältere Stände ansehen wollte. Dabei fragt man sich gegebenenfalls, warum und wann man eine Funktion eingeführt hat, oder man möchte auf einen älteren Stand zurückspringen, wenn man etwas kaputt gemacht hat. Genau an dieser Stelle kommen Versionsverwaltungsprogramme ins Spiel. Git ist eines dieser Programme, die nicht nur die bereits genannten Probleme lösen. Es ist Kernbestandteil des Entwicklungsprozesses, um sowohl kollaborativ im Team als auch alleine an einem Projekt zu arbeiten. Dabei ist es gleichgültig, ob man programmiert, Systeme administriert oder gar Bücher schreibt ist.

Randall Munroe beleuchtet in seinem Webcomic xkcd viele verschiedene Themen. Das hier abgedruckte xkcd-Comic zum Thema Git wurde während meiner Arbeit an der ersten Auflage dieses Buches veröffentlicht. Viele meiner Freunde und Bekannten aus dem Open-Source-Umfeld posteten das Comic in den verschiedenen sozialen Netzwerken und machten eins deutlich: Viele Leute nutzen zwar Git, wissen

aber nur grob, was dort passiert. Wenn etwas nicht wie geplant funktioniert oder man zu einem fehlerhaften Zustand im Arbeitsprojekt kommt, dann weiß man erst mal nicht weiter und fragt seinen persönlichen Git-Experten, wie den einen Kollegen, der glücklicherweise ein Git-Buch geschrieben hat.

Das Ziel dieses Buches ist nicht nur, dass Sie die gängigen Befehle erlernen, die Sie beim Arbeiten mit Git brauchen. Ich lege auch großen Wert auf die Einbindung und Anpassung des Entwicklungsprozesses. Darüber hinaus sollten Sie Git als Ganzes verstehen und nicht nur die Grundlagen, damit Sie mit einem Programm arbeiten, das Sie verstehen und bei dem bei Konflikten keine Hürden vorhanden sind.

Aufbau des Buches

Dieses Buch besteht aus insgesamt dreizehn Kapiteln, davon gehören die ersten vier Kapitel zu den Grundlagen und die übrigen acht zu den fortgeschrittenen Themen.

Das erste Kapitel führt in das Thema der Versionsverwaltung mit Git ein, um den Einsatzzweck und die Vorteile von Git zu verdeutlichen. Das zweite Kapitel behandelt die grundlegenden Git-Kommandos. Dies beinhaltet die Basis-Befehle, die für das Arbeiten mit Git notwendig sind. Im anschließenden dritten Kapitel geht es um die Nutzung von Branches, eines der elementaren Features von Git. So lernen Sie, mit Branches parallele Entwicklungslinien zu erstellen, zwischen diesen verschiedenen Branches hin und her zu wechseln und sie wieder zusammenzuführen. Der Grundlagenteil endet mit dem vierten Kapitel, bei dem es um den Einsatz von verteilten Repositorys geht, die es

ermöglichen, mit Repositorys zu arbeiten, die auf entfernten Servern, wie etwa GitHub oder GitLab, liegen.

Bei den fortgeschrittenen Themen liegt der Fokus besonders auf dem Einsatz von Git in Software-Entwicklungsteams. Wichtig ist dabei, über eine gute Möglichkeit zu verfügen, Git-Repositorys hosten zu können, damit man kollaborativ in einem Team an Projekten arbeiten kann. Während die wohl gängigste, bekannteste und einfachste Hosting-Möglichkeit GitHub ist, gibt es auch einige Open-Source-Alternativen, wie zum Beispiel GitLab, die sich ebenfalls sehr gut für den Einsatz in Firmen oder anderen Projektgruppen eignen. Das ist das Thema im fünften Kapitel, in dem auch der Workflow bei GitHub und GitLab thematisiert wird. Im anschließenden sechsten Kapitel geht es um die verschiedenen existierenden Workflows. Um die Features von Git sinnvoll einzusetzen, sollten Sie einen Workflow nutzen, der sowohl praktikabel ist als auch nicht zu viel Overhead im Projekt führt. Die Art und Weise, mit Git zu arbeiten, unterscheidet sich vor allem bei der Anzahl der Personen, Branches und Repositorys. Im sechsten Kapitel geht es im Anschluss darum, Git-Hooks zu verwenden, um mehr aus dem Projekt herauszuholen oder simple Fehler automatisiert zu überprüfen und somit zu vermeiden. So lernen Sie, was Hooks sind, wie sie programmiert werden und damit zu automatisieren. Generell ist dieses Kapitel für den Git-Nutzer kein alltägliches Thema. Hooks werden im Alltag eher unregelmäßig programmiert.

Die weiteren drei Kapitel befassen sich mit dem Umstieg von Subversion nach Git, wobei sowohl die Übernahme des Quellcodes inklusive der Historie als auch die Anpassung des Workflows thematisiert wird. Das neunte Kapitel ist eine Sammlung vieler verschiedener nützlicher Tipps, die zwar nicht zwangsläufig täglich gebraucht werden, aber trotzdem sehr nützlich sein können. Im zehnten Kapitel folgt dann

noch ein Kapitel mit einem Überblick über die grafischen Git-Programme unter den verschiedenen Betriebssystemen Windows, macOS und Linux. In der zweiten Auflage sind die vergleichsweise kurzen Kapitel 11 und 13 neu dazugekommen. Hier werden zum einen nützliche Hilfestellungen gegeben, um eine möglichst nachvollziehbare Git-Historie zu erzeugen, und zum anderen werden häufige Probleme von Anfängern und Erfahrenen beleuchtet und die dazugehörigen Lösungen aufgezeigt. Neu in der dritten Auflage ist das 12. Kapitel. Hier wird das Thema DevOps kurz und kompakt zusammengefasst, wofür Git das grundlegende Werkzeug ist.

Um den Einsatz von Git und die einzelnen Funktionen sinnvoll nachvollziehen zu können, werden alle Git-Kommandos anhand eines realen Beispiels erläutert. Über die Kapitel des Buches hinweg entsteht eine kleine statische Webseite, an der die Funktionen verdeutlicht werden. Denn was bringt es, die Kommandos von Git ohne den Bezug zu realen Projekten und dessen Einsatzzwecke zu kennen? Eine kleine Webseite hat insbesondere den Vorteil, dass Sie nicht nur Unterschiede im Quellcode nachvollziehen, sondern auch sehr einfach die optischen Unterschiede auf einer Webseite erkennen können.

Konvention

In diesem Buch finden Sie zahlreiche Terminal-Ausgaben abgedruckt. Diese sind größtenteils vollständig, einige mussten aus Platz- und Relevanz-Gründen jedoch gekürzt werden. Eingaben in der Kommandozeile fangen immer mit dem »$« an. Dahinter folgt dann der eigentliche Befehl. Das Dollarzeichen ist der Prompt, der in der Shell dargestellt

wird, und muss daher nicht eingetippt werden. Zeilen, die kein solches Zeichen besitzen, sind Ausgaben der Befehle. Das sieht dann etwa so aus:

```
$ git log
commit 9534d7866972d07c97ad284ba38fe84893376e20
[...]
```

Zeilen, die nicht relevant sind oder verkürzt wurden, sind als »`[...]`« dargestellt.

Hinweise und Tipps

Die einzelnen Kapitel bauen zwar aufeinander auf, doch ist es nicht immer möglich, alle Themen an Ort und Stelle ausführlich zu behandeln. Zudem werden wohl eher wenige Leser das Buch von vorne bis hinten durcharbeiten. Das Buch beinhaltet daher einige Hinweise und Tipps. Teilweise sind es Hinweise auf nähere Details in anderen Teilen des Buches, teilweise Tipps und Warnungen für die Nutzung von Git. Dies sind häufig nützliche Inhalte, die sich auf das gerade behandelte Thema beziehen, hin und wieder aber auch Querverweise zu näheren Erläuterungen in anderen Kapiteln.

Feedback

Als Autor habe ich sehr wohl den Anspruch, dass Sie als Leser das, was in diesem Buch behandelt wird, sowohl richtig verstehen als auch anwenden können. Ich bin daher

offen für Feedback und Verbesserungsvorschläge – entweder per E-Mail an mail@svij.org oder Kurzes gerne auch via Twitter an @svijee (https://twitter.com/svijee). Ich bin sehr an Ihrem Feedback interessiert!

Danksagung

Ich freue mich, dass ich erneut die Möglichkeit vom Verlag erhalten habe, dieses Buch in der nun dritten aktualisierten Auflage veröffentlichen zu dürfen. Mein Dank gilt daher erneut dem Verlag mitp und insbesondere meiner Lektorin Sabine, mit der ich nun mittlerweile fünf Jahre an diesem Buch zusammenarbeite.

Weiterhin gilt mein Dank auch dieses Mal meiner Familie und allen, die mir immer wieder neuen kleinen und großen Input und Feedback liefern.

---

[1] »xkcd: Git«, Copyright Randall Munroe (https://xkcd.com/1597/) ist lizenziert unter der Creative Commons Lizenz CC BY-NC 2.5 (https://creativecommons.org/licenses/by-nc/2.5/)

Kapitel 1
Einführung

Versionsverwaltung – Was ist denn nun eigentlich genau ein Versionsverwaltungsprogramm? Wodurch zeichnet es sich aus und warum wird es gebraucht? Das sind einige der häufigen ersten Fragen, die zu Beginn aufkommen. Die prinzipielle Bedeutung leitet sich schon aus dem Wort selbst ab: Es handelt sich um die Verwaltung von Versionen. Konkret bedeutet es, dass Sie von Dateien Versionen erzeugen können, die dann sinnvoll verwaltet werden.

Das Wort »Version« klingt zunächst erst einmal nach einer größeren Änderung, doch auch eine kleine Änderung erzeugt eine neue Version einer Datei. Je nach Kontext gibt es ein unterschiedliches Verständnis für den Begriff »Version«. Wenn bei Git von Versionen gesprochen wird, ist damit so gut wie immer die Version einer einzelnen Datei oder einer Sammlung von Dateien gemeint. Im Sinne der Software-Entwicklung werden neue Versionen von Programmen veröffentlicht, also zum Beispiel die Git-Version 2.29.

Aber wofür brauchen Sie nun ein Versionsverwaltungsprogramm wie Git? Viele kennen vermutlich folgendes Problem: Sie gehen einer Tätigkeit nach – sei es das Schreiben an einem Text, das Bearbeiten eines Bildes oder eines Videos – und der aktuelle Stand soll immer mal wieder zwischengespeichert werden. Hauptgrund ist, dass dauernd eine Sicherung der Datei vorhanden sein soll, und ein weiterer Grund ist, dass Sie wieder auf einen älteren Stand zurückspringen können, falls Sie doch einige Schritte rückgängig machen wollen. Die Vorgehensweise zum manuellen Erzeugen solcher Versionen

ist unterschiedlich – die einen fügen Zahlen mit Versionsnummern am Ende des Dateinamens an, die anderen erzeugen wiederum Ordner mit dem aktuellen Datum, in denen die Dateien liegen. So passiert es häufiger, dass neben `Bachelorarbeit_v1.odt` und `Bachelorarbeit_v2.odt` noch ein `Bachelorarbeit_v3_final.odt` und `Bachelorarbeit_v3_final_new.odt` liegt. Beide genannten Möglichkeiten funktionieren zwar prinzipiell, sind allerdings weder praktikabel noch wirklich sicher und vor allem fehleranfällig. Das ist besonders dann der Fall, wenn Sie den Dateien keine eindeutigen Namen gegeben haben. Dies trifft insbesondere dann zu, wenn zu viele Versionen einer einzigen Datei rumliegen oder mehrere Dateien gleichzeitig versioniert werden müssen.

Genau bei diesem Problem kommen Versionsverwaltungsprogramme zum Einsatz. Mit diesen werden neben den reinen Veränderungen noch weitere Informationen zu einer Version gespeichert. Darunter fallen in der Regel der Autorenname, die Uhrzeit der Änderung und eine Änderungsnotiz. Diese werden bei jeder neuen Version gespeichert. Durch die gesammelten Daten können Sie so schnell und einfach eine Änderungshistorie ansehen und verwalten. Falls zwischendurch Fehler in den versionierten Dateien eingeflossen sind, können Sie leicht untersuchen, wann und durch welche Person die Fehler eingeführt wurden, und diese wieder rückgängig machen. Versionsverwaltungsprogramme lassen sich demnach nicht nur von einzelnen Personen nutzen, sondern ermöglichen das Arbeiten im Team mit mehr als einer Person.

Mit Versionsverwaltungsprogrammen lassen sich alle möglichen Dateitypen verwalten. Sie sollten allerdings beachten, dass eine Versionierung nicht für jeden Dateityp praktikabel ist. Besonders hilfreich sind solche Anwendungen vor allem für Arbeiten mit reinen Text-

Dateien. Darunter fallen insbesondere Quellcode von Programmen, Konfigurationsdateien oder auch Texte und somit auch Bücher. Der Vorteil bei reinen Textdateien ist, dass Sie die Unterschiede bei Änderungen für jede Zeile nachvollziehen können – das ist bei binären Dateiformaten nicht möglich. Auch für Grafiker kann der Einsatz eines Versionsverwaltungsprogramms sinnvoll sein, denn mit zusätzlichen Tools können auch die Veränderungen zwischen zwei Versionen von Bildern dargestellt werden.

Insgesamt gibt es drei verschiedene Konzepte zur Versionsverwaltung: die lokale, die zentrale und die verteilte Versionsverwaltung.

1.1 Lokale Versionsverwaltung