Sujeevan Vijayakumaran

Git Schnelleinstieg

Versionsverwaltung lernen in 14 Tagen

Einfach und ohne Vorkenntnisse

Zahlreiche Praxisbeispiele

mitp

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Sujeevan Vijayakumaran

Git

Schnelleinstieg

Versionsverwaltung lernen in 14 Tagen
Einfach und ohne Vorkenntnisse

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

ISBN 978-3-7475-0527-4
1. Auflage 2022

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

© 2022 mitp Verlags GmbH & Co. KG, Frechen

Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.
Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Satz: Petra Kleinwegen

Inhalt

Einleitung

Git lernen in 14 Tagen 9
Der Aufbau des Buches 9
Konvention 10
Fragen und Feedback 10

Einführung in die Welt der Versionsverwaltung

1.1 Was ist eigentlich Versionsverwaltung? 12
1.2 Git installieren 18

Die ersten Schritte mit Git

2.1 Das erste Repository 21
2.2 Git-Konfiguration 23
2.3 Der erste Commit 24
2.4 Änderungen rückgängig machen mit Reset und Revert 42
2.5 Wie Git arbeitet 47
2.6 Git-Hilfe 52

Branches erstellen und mergen

3.1 Allgemeines zum Branching 55
3.2 Branches anlegen 57
3.3 Branches mergen 64
3.4 Merge-Konflikte 69
3.5 Verschiedene Merge-Strategien verstehen 73

Branches per Rebase zusammenführen und das Aufräumen des Repositorys

4.1 Branches per Rebase zusammenführen 75
4.2 Aufräumen mit Stash und Clean 82

Mit verteilten Repositorys arbeiten

5.1 Projekt mit einem Remote-Repository 92
5.2 Branch-Management 101
5.3 Tracking-Branches 103
5.4 Projekt mit drei Remote-Repositorys 106
5.5 Der Workflow mit drei Repositorys 109

Git-Hosting mit GitHub und GitLab

6.1 Allgemeines zum Git-Hosting 115
6.2 Einstieg in GitHub 118
6.3 Einstieg in GitLab 146
6.4 Weitere Git-Hosting-Lösungen 152

CI/CD: Continuous Integration und Continuous Delivery

7.1 Der Workflow 154
7.2 GitHub Actions 156
7.3 GitLab CI/CD 159

Workflow für Einzelpersonen

8.1 Interaktives Rebasing 166
8.2 Workflow mit einem Branch und Repository für eine Person 179

Workflows im Team

9.1 Workflow mit mehreren Personen, einem Repository und einem Branch 183
9.2 Git Flow 185
9.3 Git Flow mit mehr als einem develop-Branch 193
9.4 Git Flow mit mehreren Repositorys 195
9.5 GitHub-Flow 196
9.6 GitLab-Flow 197
9.7 Weitere Aspekte in Workflows 199

Umstieg von Subversion

10.1 Der Unterschied zwischen zentraler und verteilter Versionsverwaltung 203
10.2 Checkout und Clone im Vergleich 204
10.3 svn commit vs. git commit & git push 204
10.4 svn add vs. git add 205
10.5 Binärdateien im Repository 205
10.6 SVN- in Git-Repository konvertieren 206

Nachvollziehbare Git-Historien

11.1 Gut dosierte Commits 214
11.2 Gute Commit-Messages 215

Tipps und Tricks

12.1 Große Dateien mit Git LFS verwalten 223
12.2 Aliasse setzen mit git alias 225
12.3 Durchgeführte Aktionen nachvollziehen mit git reflog 226
12.4 Den Schuldigen finden mit git blame 229

12.5 Auf Fehlersuche mit git bisect 230
12.6 Repositorys in Repositorys mit git submodules 232
12.7 Subtree als Alternative für Submodule 236
12.8 Komplette Historie neu schreiben mit git filter-repo 238
12.9 Git Worktree 238

Frequently Asked Questions

Ausblick

14.1 Git mit GUI nutzen 247
14.2 Hooks 248
14.3 GitHub vs. GitLab 249

Stichwortverzeichnis

Einleitung

Git lernen in 14 Tagen

Mit diesem Buch haben Sie sich für einen einfachen, praktischen und fundierten Einstieg in die Welt der Versionsverwaltung mit Git entschieden. Sie lernen ohne unnötigen Ballast alles, was Sie wissen müssen, um Git effektiv für Projekte in Ihrem Berufs- und Interessensgebiet einzusetzen.

Wenn Sie Zeit genug haben, können Sie jeden Tag ein neues Kapitel durcharbeiten und so innerhalb von zwei Wochen Git lernen. Alle Erklärungen sind leicht verständlich formuliert und setzen keine Vorkenntnisse voraus. Am besten lesen Sie das Buch neben der Computer-Tastatur und probieren die Anweisungen und Befehle gleich aus und bauen sich das Beispielprojekt mit auf. Sie werden schnell erste Erfolge erzielen und den Einsatz von Git zügig verstehen können.

Der Aufbau des Buches

Das Buch beginnt mit den Grundlagen: Was ist eine Versionsverwaltung, welche Arten von Versionsverwaltungen gibt es und wie installiert man Git. Es folgen Kapitel, die die ersten Schritte in Git anhand eines Beispielprojekts einer Webseite zeigen. Die Kapitel bauen aufeinander auf. Sie lernen Schritt für Schritt, wie man Commits erstellt, Branches erstellt und die Änderungen per Merge und Rebase wieder zurückführt.. Weiterhin lernen Sie, wie sie mit verteilten Repositorys arbeiten, sei es auf GitHub oder GitLab. Wichtig und für da allgmeine Verständnis sind vor allem die Kapitel zu den Workflows: Hier lernen Sie, wie Sie Git für sich selbst und im Team erfolgreich einsetzen. Weiterhin folgt ein Kapitel zum Umstieg von Subversion, gefolgt von weiteren Kapitel zu nachvollziehbaren Git-Historien, sowie Tipps und Tricks. Das Buch schließt hab mit häufig gestellten Fragenund einem Ausblick.

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. Am Ende des Buches finden Sie ein Stichwortverzeichnis, das Ihnen hilft, bestimmte Themen im Buch schneller zu finden.

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.

Fragen und Feedback

Unsere Verlagsprodukte werden mit großer Sorgfalt erstellt. Sollten Sie trotzdem einen Fehler bemerken oder eine andere Anmerkung zum Buch haben, freuen wir uns über eine direkte Rückmeldung an *lektorat@mitp.de*.

Falls es zu diesem Buch bereits eine Errata-Liste gibt, finden Sie diese unter *www.mitp.de/0526* im Reiter Downloads.

Wir wünschen Ihnen viel Erfolg und Spaß beim Lernen von Git!

Sujeevan Vijayakumaran und das mitp-Lektorat

Einführung in die Welt der Versionsverwaltung

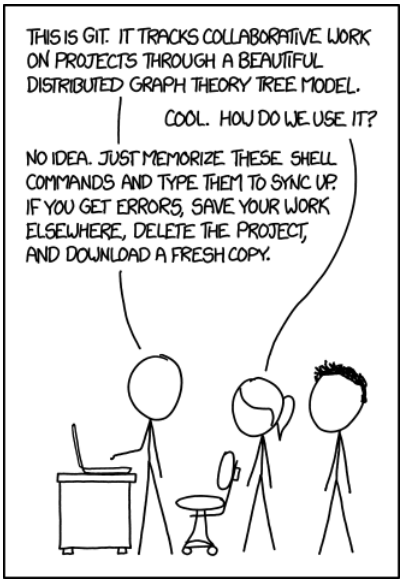

Abb. 1.1: »Das ist Git. Es bietet einen Überblick über die kollaborative Arbeit in Projekten durch die Nutzung eines wunderschönen Graphen-Theorie-Modells.«

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 hin, 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 von »Es ist wirklich gar nicht so schwer, stell dir nur die Branches als …“, und irgendwann lernst du die Befehle, die jedes Problem fixen.

»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/*)

Versionskontrolle ist ein wichtiges Thema für Software-Entwickler. Jeder, der ohne jegliche Versionskontrollprogramme arbeitet, ist vermutlich schon einmal an den Punkt gestoßen, an dem 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.

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.

1.1 Was ist eigentlich Versionsverwaltung?

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 Da-

tei. 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 insbesonde-

re 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.

1.1.1 Lokale Versionsverwaltung

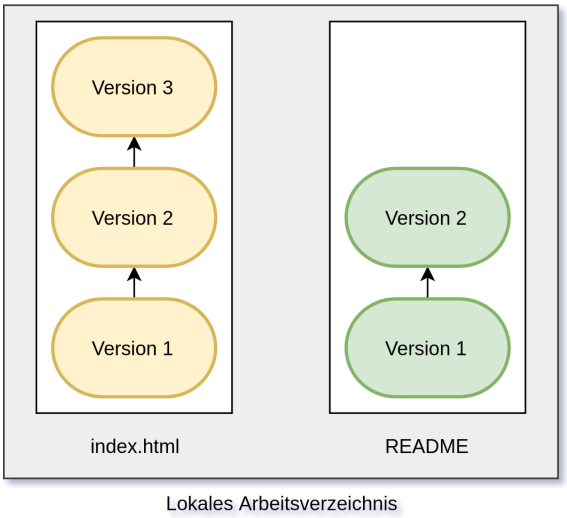

Abb. 1.2: Lokale Versionsverwaltung arbeitet dateibasiert und lediglich lokal.

Die lokale Versionsverwaltung findet sich eher seltener in produktiven Umgebungen, da sie lediglich lokal arbeitet und häufig nur einzelne Dateien versioniert. Die zuvor erwähnte manuelle Erzeugung von Versionen von Dateien wäre zum Beispiel eine lokale Versionsverwaltung mit einer einzelnen Datei. Sie ist zwar einfach zu nutzen, doch ist es fehleranfällig und wenig flexibel. Echte Versionsverwaltungssoftware, die nur lokal arbeitet, gibt es allerdings auch, darunter »SCSS« und »RCS«. Der größte Nachteil lokaler Versionsverwaltung ist, dass im Normalfall nur eine Person mit den Dateien arbeiten kann, da diese nur lokal auf dem einen Gerät verfügbar sind. Weiterhin be-

steht keine Datensicherheit, da die Dateien nicht automatisch auf einem anderen Gerät gesichert werden. Der Anwender ist somit allein verantwortlich für ein Backup der Dateien inklusive der Versionshistorie.

1.1.2 Zentrale Versionsverwaltung

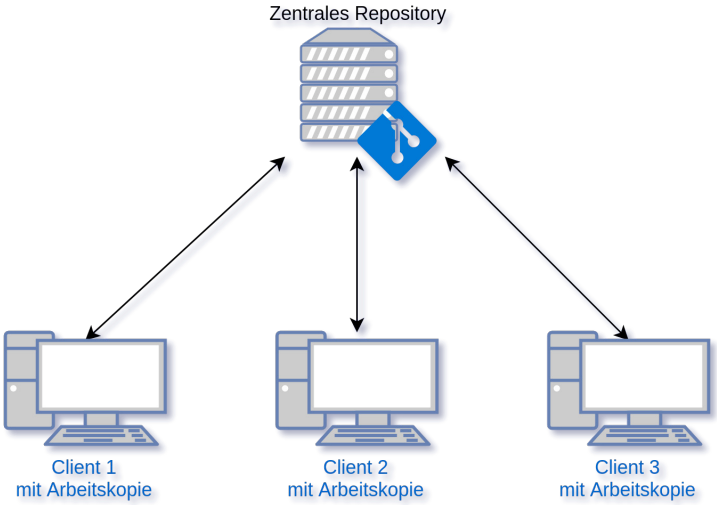

Abb. 1.3: Zentrale Versionsverwaltung arbeitet mit Arbeitskopien auf Clients.

Zentrale Versionsverwaltungen befinden sich heute vergleichsweise noch häufig im Einsatz. Bekannte und verbreitete Vertreter dieser Art sind Subversion und CVS. Das Hauptmerkmal zentraler Versionsverwaltungen ist, dass das Repository lediglich auf einem zentralen Server liegt. Das Wort »Repository« ist Englisch und steht für »Lager«, »Depot« oder auch »Quelle«. Ein Repository ist somit ein Lager, in dem die versionierten Dateien liegen. Autorisierte Nutzer verfügen über eine lokale Arbeitskopie einer Version, auf der sie ihre Arbeiten erledigen.

Die Logik und die Daten der Versionsverwaltung liegen größtenteils auf dem zentralen Server. Beim Wechsel von Revisionen oder beim Vergleichen von Änderungen wird stets mit dem Server kommuniziert. Wenn der Server also offline ist, kann der Nutzer zwar mit der Arbeitskopie ein wenig weiterarbeiten. Allerdings ist die Einsicht älterer Versionen oder das Ansehen anderer Entwicklungslinien nicht möglich, da es sich lediglich um eine Arbeitskopie einer Version und keine Kopie des vollständigen Repositorys handelt.

1.1.3 Verteilte Versionsverwaltung

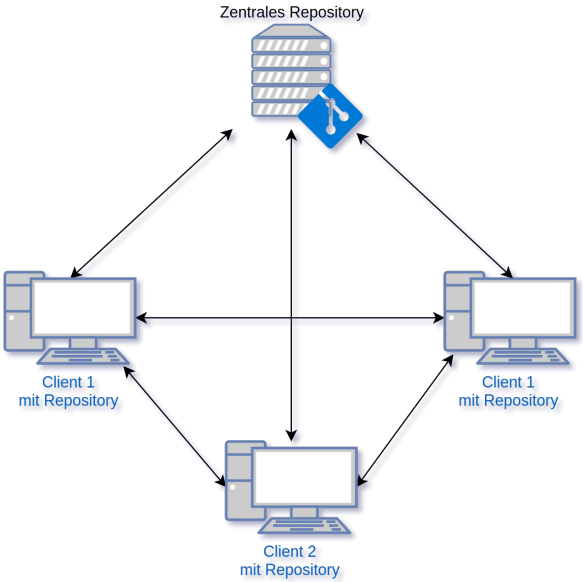

Abb. 1.4: Verteilte Versionsverwaltung arbeitet mit Repositorys auf Clients und Servern.

Git gehört zu den verteilt arbeitenden Versionsverwaltungsprogrammen. Neben Git gibt es auch andere verteilte Versionskontrollprogramme, wie Bazaar oder Mercurial. Im Gegensatz zur zentralen Versionsverwaltung besitzt jeder Nutzer des Repositorys nicht nur eine Arbeitskopie, sondern das komplette Repository. Wenn Sie also zwischen verschiedenen Revisionen wechseln oder sich die Historie einzelner Dateien anschauen möchte, dann geschieht das Ganze auf dem lokalen Rechner. Zuvor muss nur das Repository »geklont« werden. Alle Funktionen stehen dann auch offline zur Verfügung. Ein wesentlicher Vorteil davon ist, dass nicht nur unnötiger Datenverkehr vermieden wird, sondern auch die Geschwindigkeit deutlich höher ist, was durch die fehlende Netzwerklatenz bedingt ist.

Zusätzlich besitzen verteilte Versionsverwaltungssysteme eine höhere Datenausfallsicherheit, da die Kopien der Daten des Repositorys in der Regel auf verschiedenen Rechnern liegen. Bei einem Ausfall des Git-Servers ist es daher möglich, weiterzuarbeiten. Nichtsdestotrotz sollten Sie von wichtigen Daten natürlich immer Backups anfertigen, ganz egal ob es sich um lokale, zentrale oder verteilte Versionsverwaltung handelt.

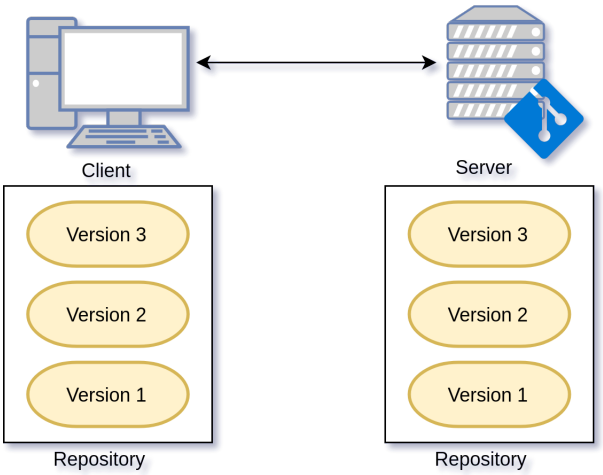

Abb. 1.5: Die Versionshistorie liegt sowohl lokal auf dem Client als auch auf dem Server.

Um den Unterschied zwischen zentralen und verteilten Versionsverwaltungsprogrammen klarer zu machen, kann folgendes Beispiel helfen. Stellen Sie sich vor, dass das Repository ein dicker Aktenordner ist. Darin enthalten sind alle aktuellen Dateien, ältere Versionen der Dateien sowie die Änderungshistorie mitsamt den Kommentaren zu den Änderungen. Sie müssen mit diesen Dateien arbeiten. Wenn es sich um ein zentrales System handelt, dann befindet sich der Aktenordner an einer zentral zugänglichen Stelle, die hier nun Archiv genannt wird. Für Sie heißt es, dass Sie zum Archiv und zu dem Ordner gehen müssen. Dort wird dann eine Arbeitskopie der benötigten Dateien erzeugt und anschließend laufen Sie wieder zurück zum Arbeitsplatz. Wenn Sie die Änderungshistorie von einer oder mehreren Dateien ansehen möchten, müssen Sie immer wieder zum Archiv laufen und den Aktenordner durchblättern, um sich diese anzusehen. Da es sowohl Zeit als auch Energie kostet, immer zum zentralen Aktenordner zu laufen, bietet es sich an, eine Kopie des ganzen Ordners zu erstellen und mit an Ihren Arbeitsplatz zu nehmen.

Genau das ist dann eine verteilte Versionsverwaltung, da nun zwei vollständige Kopien des Aktenordners existieren – einmal an zentraler Stelle im Archiv und einmal am eigenen Arbeitsplatz. Der Vorteil ist, dass nach der ersten Kopie nur noch die Veränderungen hin- und hergetragen werden müssen. Alles andere kann bequem vom Arbeitsplatz aus gemacht werden, ohne ständig aufzustehen und herumlaufen zu müssen. Konkret bedeutet das, dass Sie an Ihrem Arbeitsplatz sitzen und Ihre Aufgaben erledigen. Sobald die Arbeit ab-

geschlossen ist, tragen Sie nur die neuen Dateien zum Archiv, wo Sie eine Kopie anfertigen und diese im zentralen Aktenordner abheften. Großer Vorteil ist, dass Sie auch weiterhin arbeiten können, wenn der Weg zum Aktenordner unzugänglich ist, etwa genau dann, wenn Sie unterwegs sind.

Zusammenfassung

- Die lokale Versionsverwaltung funktioniert lediglich auf einem einzelnen Rechner.
- Bei der zentralen Versionsverwaltung liegt das »Gehirn« auf einem zentralen Server, von dem sich alle Mitarbeiter eine Arbeitskopie ziehen können.
- Bei der verteilten Versionsverwaltung liegt das vollständige Repository sowohl auf mindestens einem Server sowie auf allen Clients, wo mit Klonen gearbeitet wird.

1.2 Git installieren

Bevor Sie losgelegen, müssen Sie Git installieren. Git gibt es nicht nur für die gängigen Betriebssysteme Windows, macOS und Linux, sondern unter anderem auch für FreeBSD, Solaris und sogar Haiku. Die gängigen Linux-Distributionen stellen Git unter dem Paketnamen »git« in der Paketverwaltung zur Verfügung. Nutzer von Windows und macOS können sich Git von der Projektwebsite *https://git-scm.com/downloads* herunterladen.

Während der Arbeit an diesem Buch im Januar 2022 ist die Git-Version 2.34 die neueste Version. Große Unterschiede zu den vorherigen Versionen seit 2.0 existieren hingegen nicht, es sind vielmehr zahlreiche Kleinigkeiten, die über die Zeit eingeflossen sind. Bei Bedarf werden neue Funktionen aus vergleichsweise neuen Versionen hervorgehoben. Gleiches gilt für möglicherweise ältere Versionen von Git, die sich noch in den Paketverwaltungen älterer Linux-Distributionen finden. Obwohl die Version 2.0 von Git schon über acht Jahre alt ist, werden an den ein oder anderen Stellen im Buch noch Unterschiede zu den mittlerweile sehr alten Versionen hervorgehoben. Als Neuankömmling sehen Sie so, was sich getan hat, und wenn Sie nach etlichen Jahren Pausen dann doch wieder Git anfassen, dann geht Ihnen auch nichts verloren.

Die Installation unter Windows ist größtenteils selbsterklärend. So können Sie die Standardeinstellungen beibehalten und bei der Installation entsprechend durchklicken. Etwaig aufploppende Abfragen zu Standardeinstellungen

können und werden später im Buch behandelt wie etwa der standardmäßige Branchname beim Anlegen eines Repositorys.

Bei der Nutzung von Git präferiere ich die Git-Bash, da sie im Defaultzustand einige zusätzliche Funktionen bietet, wie die Anzeige des Namens des aktuellen Branches. Außerdem können die gängigen Unix-Kommandos verwendet werden. Alle Befehle in diesem Buch lassen sich problemlos in einer Shell unter macOS und Linux bzw. der Git-Bash unter Windows ausführen. Die Windows-Cmd kann zwar auch verwendet werden, allerdings nenne ich Windows-Cmd-Kommandos nicht noch einmal explizit. Dies ist unter anderem dann relevant, wenn etwa Ordner angelegt oder Dateien verschoben werden sollen.

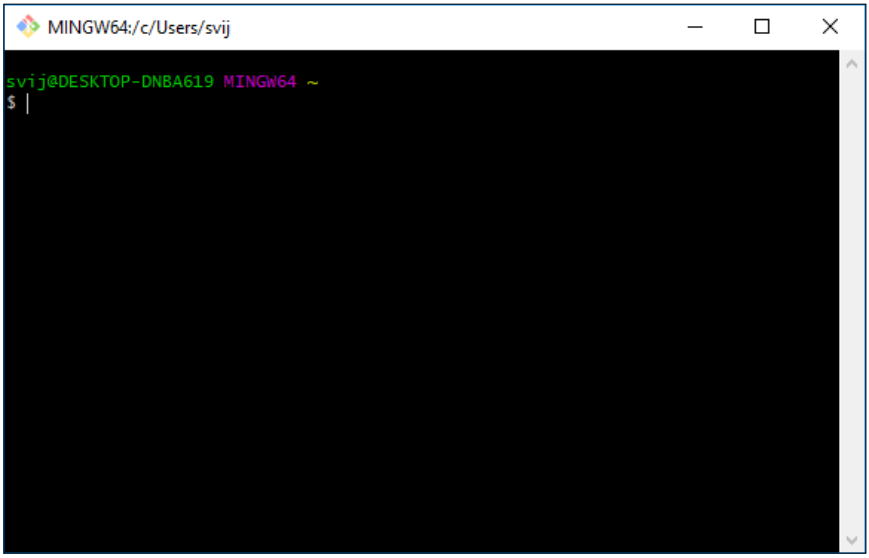

Abb. 1.6: Git-Bash unter Windows

2 Die ersten Schritte mit Git

In diesem Kapitel lernen Sie die grundlegenden Funktionen und Kommandos von Git kennen. So gut wie alle in diesem Kapitel behandelten Befehle dürften beim täglichen Arbeiten mit Git zum Einsatz kommen. Damit Sie den Sinn und Zweck einzelner Befehle und Funktionen von Git sowohl nutzen als auch nachvollziehen können, arbeiten Sie in diesem Kapitel hauptsächlich an einem Beispielprojekt, das die Nutzung und Arbeitsweise von und mit Git verdeutlicht.

Das Beispiel-Projekt ist eine kleine Website, die nach und nach aufgebaut wird. HTML-Kenntnisse sind prinzipiell nicht notwendig, können aber natürlich auch nicht schaden. Damit es nicht ganz so trocken und langweilig ist, sollten Sie die Beispiele auf dem eigenen Rechner auf jeden Fall nachmachen. An der ein oder anderen Stelle bietet es sich es auch an, etwas herumzuexperimentieren, denn nur durch Praxis wird Ihnen das Arbeiten mit Git klar und Sie haben hinterher in echten Projekten keine großen Probleme.

Als Beispiel wird eine kleine persönliche Website mit dem HTML5-Framework »Bootstrap« erstellt. Auf die genaue Funktionsweise des Frameworks gehe ich nicht näher ein, da es sich hier ja um Git und nicht um HTML und CSS dreht.

Git ist traditionell ein Kommandozeilenprogramm, weshalb der Fokus auf der Arbeit mit Git in der Kommandozeile liegt.

2.1 Das erste Repository

Da Git ein verteiltes Versionsverwaltungsprogramm ist, lassen sich alle Operationen an einem Repository vollständig lokal ausführen. Alle Git-Funktionen, die in diesem Kapitel erläutert werden, sind ohne Ausnahme lokale Befehle

auf dem eigenen Rechner. Verbindungen zu externen Git-Servern werden also nicht aufgenommen.

Bevor Sie die ersten Dateien in ein Repository schieben können, müssen sie lokal angelegt werden. Da Git ein Kommandozeilenprogramm ist, müssen Sie jetzt unter Linux oder macOS das Terminal öffnen. Windows-Nutzer rufen an dieser Stelle die Git-Bash auf.

Im Defaultzustand landet man dann im Home-Verzeichnis des Nutzerkontos. Dies ist etwa `/home/svij` unter Linux, `C:\Users\svij` unter Windows oder `/Users/svij` unter macOS. In diesem oder in einem anderen beliebigen Verzeichnis legen Sie nun einen Unterordner namens `meineWebsite` an, in dem das Repository und dessen Daten liegen sollen. Anschließend wechseln Sie mit dem `cd`-Befehl in das Verzeichnis.

```
$ mkdir meineWebsite
$ cd meineWebsite
```

In diesem Verzeichnis soll nun das Git-Repository angelegt werden. Dazu reicht ein einfaches Ausführen des folgenden Befehls:

```
$ git init
Hinweis: Als Name für den initialen Branch wurde 'master'
Hinweis: benutzt. Dieser Standard-Branchname kann sich ändern. Um
Hinweis: den Namen des initialen Branches zu konfigurieren, der
Hinweis: in allen neuen Repositories verwendet werden soll und um
Hinweis: diese Warnung zu unterdrücken, führen Sie aus:
Hinweis:
Hinweis:        git config --global init.defaultBranch <Name>
Hinweis:
Hinweis: Häufig gewählte Namen statt 'master' sind 'main', 'trunk'
Hinweis: und 'development'. Der gerade erstellte Branch kann mit
Hinweis: diesem Befehl umbenannt werden:
Hinweis:
Hinweis:        git branch -m <Name>
Hinweis: Leeres Git-Repository in /home/sujee/Repositorys/
Hinweis: meineWebsite/.git/ initialisiert
```

Die Ausgabe des Befehls verrät schon, was passiert ist. Git hat innerhalb des Projekt-Ordners ein `.git`-Verzeichnis angelegt, in dem das leere Git-Repository liegt. Es liegen zwar Dateien und Verzeichnisse im `.git`-Verzeichnis, doch ist das Repository prinzipiell leer, da noch keine Daten und keine Revisionen

hinterlegt sind. Das Verzeichnis ist auf allen Betriebssystemen versteckt. Das Gedächtnis des Git-Repositorys liegt vollständig im `.git`-Unterverzeichnis. Falls man das Verzeichnis löscht, sind auch alle gespeicherten Informationen des Repositorys gelöscht. Zum jetzigen Zeitpunkt wäre das natürlich nicht so tragisch, da es noch leer ist.

In der Version Git 2.30, die Anfang 2021 erschien, wurden die oben aufgeführten Hinweise ergänzt, damit im Standard Repositorys beim Erstellen nicht mit dem `master` Branch angelegt werden müssen. Für dieses Buch belasse ich es zunächst bei der Nutzung des `master` Branches.

Was Branches sind und was die Namen genau zu bedeuten haben, folgt sowieso noch an späterer Stelle.

An dieser Stelle lohnt sich schon ein kleiner Blick in dieses Verzeichnis:

```
$ ls -l .git
insgesamt 12
drwxr-xr-x 1 sujee sujee   0 30. Dez 20:10 branches
-rw-r--r-- 1 sujee sujee  92 30. Dez 20:10 config
-rw-r--r-- 1 sujee sujee  73 30. Dez 20:10 description
-rw-r--r-- 1 sujee sujee  23 30. Dez 20:10 HEAD
drwxr-xr-x 1 sujee sujee 414 30. Dez 20:10 hooks
drwxr-xr-x 1 sujee sujee  14 30. Dez 20:10 info
drwxr-xr-x 1 sujee sujee  16 30. Dez 20:10 objects
drwxr-xr-x 1 sujee sujee  18 30. Dez 20:10 refs
```

Wie Sie sehen, liegen in dem `.git`-Verzeichnis einige Verzeichnisse und Dateien. Was genau darin passiert, ist an dieser Stelle zunächst irrelevant. Händisch muss in diesem Verzeichnis in der Regel zunächst nichts unternommen werden, außer wenn Sie Hooks – für das Ausführen von Skripten bei diversen Aktionen – oder die Konfiguration anpassen möchten. Allerdings sollten Sie die Dateien nur anfassen, wenn Ihnen bekannt ist, zu welchen Auswirkungen es führt, denn sonst können Sie das Repository kaputtmachen!

2.2 Git-Konfiguration

Da Sie bereits ein leeres Repository angelegt haben, können Sie nun ein Commit hinzufügen. Was genau ein Commit ist und wie es getätigt werden kann, wird später genau erläutert. Denn zunächst müssen Sie noch die Git-Installation konfigurieren.

Vorerst werden allerdings nur zwei Dinge konfiguriert: der eigene Entwicklername und die dazugehörige E-Mail-Adresse.

Mit den folgenden Befehlen können Sie den eigenen Namen und die eigene E-Mail-Adresse setzen:

```
$ git config --global user.name "Sujeevan Vijayakumaran"
$ git config --global user.email mail@svij.org
```

An dieser Stelle sollten Sie natürlich dann Ihren Namen und E-Mail-Adresse eintragen und nicht meine Daten.

Mit diesen beiden Befehlen wird die Datei `.gitconfig` im Home-Verzeichnis angelegt. Der Inhalt der Datei in `~/.gitconfig` sieht anschließend so aus:

```
$ cat ~/.gitconfig
[user]
        name = Sujeevan Vijayakumaran
        email = mail@svij.org
```

Mit dem Befehl `git config -l` lässt sich die Konfiguration ebenfalls über die Kommandozeile ansehen.

Beachten Sie, dass bei den oben genannten Befehlen die Git-Identität global für das Benutzerkonto des Betriebssystems gesetzt wird. Wenn Sie für einzelne Git-Repositorys spezifische Einstellungen setzen möchten, reicht es, den Aufruf-Parameter `--global` wegzulassen. Die Konfiguration wird dann in die Datei `.git/config` im Projektordner gespeichert. Dies ist häufig dann sinnvoll, wenn Sie verschiedene E-Mail-Adressen für verschiedene Projekte nutzen. Das trifft beispielsweise dann zu, wenn Sie für die Erwerbsarbeit eine E-Mail-Adresse verwenden und für private Repositorys eine andere. Die angegebenen Informationen zu einem Entwickler sind für alle Personen einsehbar, die mindestens Lese-Rechte im Repository besitzen, sofern der Entwickler mindestens einen Commit getätigt hat.

2.3 Der erste Commit

An dieser Stelle startet das »echte« Arbeiten mit dem Git-Repository. Wie bereits vorher erwähnt, sind sowohl das Arbeitsverzeichnis als auch das Repository leer. Sie müssen daher einige Dateien in das Arbeitsverzeichnis schieben.

Zuvor lohnt sich noch ein Ausführen des Git-Kommandos `git status`:

```
$ git status
Auf Branch master
Noch keine Commits
nichts zu committen (erstellen/kopieren Sie Dateien und benutzen
Sie "git add" zum Versionieren)
```

Dieser Befehl gibt immer sinnvolle und praktische Informationen aus, die für das Arbeitsverzeichnis und für das Repository zu der entsprechenden Zeit hilfreich sind. Zum jetzigen Zeitpunkt teilt es mit, dass man sich auf dem Branch `master` befindet, noch keine Commits vorhanden sind und es noch nichts zu committen gibt. Es handelt sich demnach um ein noch leeres Repository.

Jetzt ist es an der Zeit, die ersten Dateien hinzuzufügen und den ersten Commit zu tätigen. Da in diesem Beispielprojekt das HTML5-Framework Bootstrap verwendet wird, müssen Sie dieses zunächst herunterladen und entpacken. Dies kann entweder händisch geschehen oder Sie führen folgende Befehle aus. Falls `curl` nicht installiert ist, was bei einigen Linux-Distributionen der Fall sein kann, können Sie es nachinstallieren, das Programm `wget` nutzen oder die Datei über den angegebenen Link händisch über den Browser herunterladen und entpacken.

```
$ curl -o bootstrap.zip -L https://github.com/twbs/bootstrap/
releases/download/v4.6.1/bootstrap-4.6.1-dist.zip
$ unzip bootstrap.zip
$ mv bootstrap-4.5.2-dist/* .
$ rmdir bootstrap-4.5.2-dist
$ rm bootstrap.zip
```

Einige der aufgeführten Befehle geben Text auf der Standard-Ausgabe aus, die ich hier aus Gründen der Übersichtlichkeit weggelassen habe. Über die Befehle wurde der Download getätigt, die Zip-Datei entpackt und somit ihr Inhalt in das Projektverzeichnis geschoben. Anschließend liegen im Projektverzeichnis dann zwei Unterverzeichnisse: `css` und `js`.

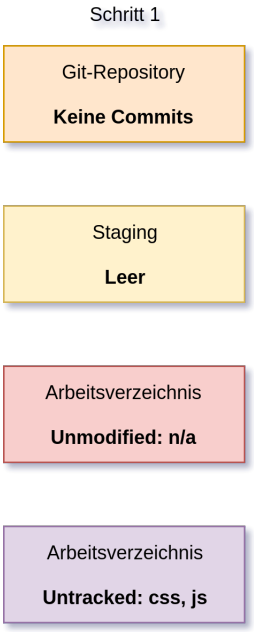

Abb. 2.1: Das Arbeitsverzeichnis ist gefüllt, Repository und Staging sind leer.

Obwohl die Dateien und Ordner im Projektordner liegen, sind die Dateien noch nicht im Repository. Sie müssen Git immer explizit mitteilen, dass Dateien in das Repository geschoben werden sollen.

Generell ist es durchaus häufig sinnvoll, den aktuellen Status im Arbeitsverzeichnis zu prüfen, deshalb lohnt sich jetzt ein Blick auf die Ausgabe von `git status`:

```
$ git status
Auf Branch master
Noch keine Commits
Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
   css/
   js/
nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
```