GANZ EINFACH

Versionskontrolle verstehen

Änderungen an Code, Texten und Projekten sicher nachverfolgen



Danilo Sieren

G A N Z E I N F A C H

Band 60

Versionskontrolle verstehen

Änderungen an Code, Texten und Projekten sicher nachverfolgen

Danilo Sieren

GANZ EINFACH

Die Reihe GANZ EINFACH erklärt Technik und Alltagsthemen so, wie es eine gute Freundin oder ein guter Freund am Küchentisch tun würde: geduldig, ehrlich und mit vielen Beispielen. Jeder Band steht für sich, und wo ein anderer Band ein Thema vertieft, steht ein Hinweis im Text.

Impressum und Hinweise zur Entstehung stehen auf der letzten Seite dieses Buches.

Vorwort

Auf fast jedem Rechner gibt es einen Ordner, in dem Dateien mit Namen wie »Angebot_neu.docx«, »Angebot_neu_korrigiert.docx« und »Angebot_FINAL_wirklich.docx« nebeneinanderliegen. Niemand weiß mehr genau, welche davon gilt. Die Version vom Dienstag hatte den besseren Einleitungssatz, die vom Donnerstag die richtigen Preise, und irgendwo dazwischen hat jemand einen Absatz gelöscht, den man jetzt dringend bräuchte. Wer so arbeitet, macht nichts falsch. Er benutzt nur das einzige Werkzeug, das ihm niemand erklärt hat: den Dateinamen.

Dieses Buch stellt dir ein besseres Werkzeug vor. **Versionskontrolle** heißt die Technik, mit der Programmiererinnen und Programmierer seit Jahrzehnten ihre Arbeit sichern. Jede Änderung wird festgehalten, mit Datum, Namen und einer kurzen Begründung. Man kann jederzeit zu einem früheren Stand zurückkehren, zwei Fassungen vergleichen und sehen, wer wann welche Zeile verändert hat. Mehrere Menschen können gleichzeitig an denselben Dateien arbeiten, ohne sich gegenseitig zu überschreiben. Das bekannteste Programm dafür heißt **Git**. Es ist kostenlos, läuft auf jedem Rechner und wird von Millionen Menschen benutzt, längst nicht nur für Programmcode.

Git hat einen Ruf, der ihm vorauseilt. Es gilt als mächtig, aber unzugänglich, mit Befehlen, die wie Zaubersprüche klingen, und Fehlermeldungen, die mehr verwirren als helfen. Ein Teil dieses Rufs ist verdient. Der größere Teil kommt daher, dass Git meistens im Vorbeigehen erklärt wird, als Liste von Befehlen zum Abschreiben. Dieses Buch geht anders vor. Es erklärt zuerst, was Versionskontrolle überhaupt leistet und wie Git darüber denkt. Wer das Modell verstanden hat, braucht keine Zaubersprüche mehr. Die Befehle ergeben sich dann fast von selbst.

Du lernst, wie du ein Projekt unter Versionskontrolle stellst, Änderungen sinnvoll festhältst, frühere Fassungen wiederherstellst und Fehler rückgängig machst. Du lernst, mit Zweigen parallel an verschiedenen Ideen zu arbeiten und sie später zusammenzuführen. Du erfährst, wie du deine Arbeit auf einen Server bringst, mit anderen teilst und gemeinsam daran arbeitest, auf großen Plattformen ebenso wie auf einem eigenen kleinen Server. Und du siehst, dass sich Versionskontrolle nicht nur für Programme eignet, sondern

ebenso für Handbücher, Abschlussarbeiten, Webseiten, Übersetzungen, Notensammlungen und Konfigurationsdateien.

Durch das Buch begleitet dich Anke Rüther. Sie ist 46, arbeitet als freiberufliche Übersetzerin und technische Redakteurin in Unna und hat gerade den größten Auftrag ihrer Laufbahn angenommen: das Handbuch für die neuen Fahrradanhänger einer Firma aus Kamen, auf Deutsch, Englisch und Niederländisch, mit regelmäßigen Aktualisierungen über drei Jahre. Ihre bisherige Arbeitsweise mit nummerierten Dateien stößt dabei schnell an ihre Grenzen. Hilfe bekommt sie von Dilara Aksoy, 31, Webentwicklerin, mit der sie sich ein Büro teilt und die Git seit dem Studium jeden Tag benutzt. Und dann ist da noch Ankes Sohn Leon, 17, der in seinem Zimmer an einem kleinen Computerspiel programmiert und bisher jeden zweiten Abend verzweifelt, weil eine Änderung alles kaputtgemacht hat, was am Vortag noch lief.

Du brauchst kein technisches Vorwissen. Wenn du Ordner anlegen, Dateien speichern und ein Programm installieren kannst, bringst du alles mit. Einige Kapitel arbeiten mit der Kommandozeile, also mit getippten Befehlen. Keine Sorge: Ich erkläre jeden Befehl, und für fast alles gibt es zusätzlich grafische Programme, die ich dir ebenfalls vorstelle.

Lege einen neuen Ordner an. Wir fangen an.

Danilo Sieren, Dortmund, im November 2026

So arbeitest du mit diesem Buch

Das Buch ist so aufgebaut, dass du es von vorn nach hinten lesen kannst. Die ersten beiden Kapitel erklären, warum Versionskontrolle hilft und welche Ideen dahinterstecken. Danach richtest du Git ein und lernst die tägliche Arbeit kennen: Änderungen festhalten, den Verlauf lesen, Fehler rückgängig machen. Im Mittelteil geht es um Zweige, das Zusammenführen von Arbeit und den Umgang mit Konflikten. Ab Kapitel 15 verlässt Git deinen Rechner: Du lernst Server, Plattformen und die Zusammenarbeit im Team kennen. Die letzten Kapitel widmen sich Texten und großen Dateien, grafischen Programmen, Sicherheit, Automatisierung, Rettungstechniken, Datensicherung und der Arbeit mit KI-Assistenten.

Wenn du nur für dich allein arbeitest und keine Programme schreibst, sind die Kapitel 1 bis 9, 14, 20 und 26 dein Kern. Lies aber auch Kapitel 23 über Sicherheit, bevor du ein Projekt zum ersten Mal auf einen Server lädst. Was dort einmal veröffentlicht wurde, lässt sich nur mit Mühe wieder zurückholen.

Die Bausteine jedes Kapitels

Jedes Kapitel beginnt mit einer Erklärung in ganz normaler Sprache. Fachbegriffe erkläre ich beim ersten Auftreten, und im Glossar in Anhang A findest du alle noch einmal. Git benutzt viele englische Wörter, für die es keine eingebürgerte deutsche Entsprechung gibt. Ich verwende sie so, wie sie in der Praxis gebraucht werden, und erkläre sie. Anhang K listet sie mit Übersetzung auf.

In der **Werkstatt** setzt du das Gelernte Schritt für Schritt um. Die Schritte sind nummeriert und so beschrieben, dass du sie direkt am Rechner nachvollziehen kannst.

Die Kästen mit der Überschrift **Vertiefung** gehen tiefer in die Technik. Beim ersten Lesen kannst du sie überspringen. **Achtung**-Kästen warnen vor Fehlern, die Daten oder Nerven kosten können. **Tipp**-Kästen enthalten Abkürzungen aus der Praxis. In den **Lernprojekt**-Kästen begleitest du Anke, Dilara und Leon.

Am Ende jedes Kapitels beantworte ich **Häufige Fragen** und fasse in **Kurz gesagt** das Wichtigste zusammen.

Wie Befehle im Buch aussehen

Befehle und Ausgaben stehen in einem grauen Kasten in einer Schrift, in der alle Zeichen gleich breit sind:

```
git status
git commit -m "Kapitel 3 übersetzt"
```

Einzelne Befehle im Fließtext sind ebenfalls in dieser Schrift gesetzt, etwa `git log`. Was du selbst eintippst, steht ohne Vorzeichen da. Wenn ich zeige, was Git darauf antwortet, steht die Antwort direkt darunter. Tippe die Befehle am besten selbst ab. Beim Abtippen merkst du dir die Schreibweise, und kleine Tippfehler sind gute Gelegenheiten, Git beim Erklären zuzusehen. Eine Übersicht der wichtigsten Befehle findest du in Anhang B.

Programme und Versionen

In diesem Buch nenne ich Programme, Plattformen und Dienste bekannter Anbieter als Beispiele. Das ist keine Werbung, sondern soll dir die Orientierung erleichtern. Alle Werkzeuge, die du brauchst, gibt es kostenlos. Das Buch beschreibt den Stand vom November 2026. Die aktuelle Git-Version ist 2.56, und die Entwickler haben angekündigt, im Frühjahr 2027 die Version 3.0 zu veröffentlichen. Sie wird einige Voreinstellungen ändern, die ich an den passenden Stellen erwähne. Die Arbeitsweise, die du hier lernst, bleibt davon unberührt.

> **ACHTUNG** **Erst üben, dann am Ernstfall**
>
> Probiere neue Befehle zuerst in einem Übungsordner aus, bevor du sie auf ein wichtiges Projekt anwendest. Die allermeisten Git-Befehle sind harmlos und lassen sich rückgängig machen. Einige wenige können aber Arbeit löschen, die noch nicht gesichert war. Ich kennzeichne sie im Text deutlich.

Inhalt

Vorwort 3
So arbeitest du mit diesem Buch 5
Warum Versionskontrolle hilft 9
Die Grundideen 16
Git installieren und einrichten 23
Das erste Repository 30
Drei Bereiche: Arbeitsordner, Staging, Repository 38
Gute Commits und gute Nachrichten 44
Die Geschichte lesen 50
Fehler rückgängig machen 56
Ordnung im Projekt: Dateien ausschließen 63
Zweige: parallel an Ideen arbeiten 70
Zusammenführen: Merge 76
Konflikte lösen 82
Geschichte aufräumen 88
Markierungen und Versionsnummern 94
Git verlässt den Rechner: entfernte Repositories 99
Sicher anmelden: Schlüssel und Zugangstoken 106
Plattformen im Vergleich 112
Gemeinsam arbeiten: Pull Requests und Reviews 119
Arbeitsabläufe für Teams 125
Texte versionieren 131
Große Dateien, Bilder und Medien 138
Grafische Programme und Editoren 144
Sicherheit und Geheimnisse 150
Automatisieren: Hooks und Prüfungen 156
Rettung in der Not 163
Datensicherung und Git 169
Versionskontrolle und KI-Assistenten 174
Wie es weitergehen kann 179
Glossar 185
Git auf einen Blick 189
Übersichten 192
Fehlermeldungen verstehen 194
Vorlagen 195
Checklisten 197
Fragen zum Selbsttest 199
Problemlöser 201

Die Stationen des Lernprojekts 203
Wichtige Adressen 205
Englische Begriffe 206
Nachwort 208
Quellen und weiterführende Links 209
Ausführliches Inhaltsverzeichnis 211
Stichwortverzeichnis 224
Impressum 229

KAPITEL 1

Warum Versionskontrolle hilft

Jeder, der mit Dateien arbeitet, betreibt bereits eine Art Versionskontrolle. Wer vor einer großen Änderung schnell eine Kopie anlegt, sichert eine Version. Wer eine Datei »Bericht_alt.docx« nennt, bevor er sie überarbeitet, markiert einen Stand. Wer in Word die Funktion »Änderungen nachverfolgen« einschaltet, protokolliert, wer was geändert hat. Das alles sind vernünftige Ideen. Sie haben nur einen gemeinsamen Haken: Sie hängen davon ab, dass man im richtigen Moment daran denkt, und sie werden mit jeder Woche unübersichtlicher.

Eine **Versionskontrolle** im technischen Sinn ist ein Programm, das diese Arbeit übernimmt. Es merkt sich jeden Stand, den du ausdrücklich festhältst, mit Datum, Uhrzeit, deinem Namen und einer kurzen Beschreibung. Es kann jeden dieser Stände wiederherstellen, zwei beliebige miteinander vergleichen und zeigen, welche Zeile seit wann so aussieht, wie sie aussieht. Und es tut das nicht für eine einzelne Datei, sondern für einen ganzen Ordner mit allen Unterordnern.

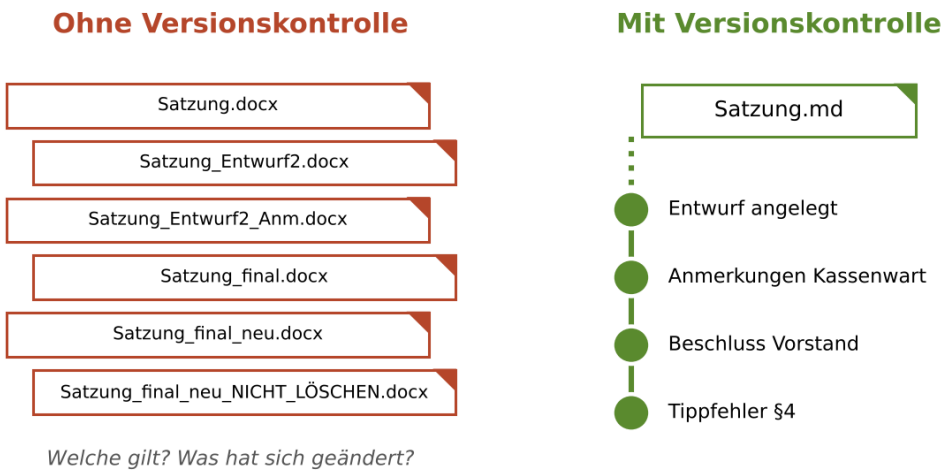
Das Problem mit »final_final«

Stell dir ein typisches Projekt vor: eine Vereinssatzung, eine Bachelorarbeit, das Handbuch einer Maschine. Am Anfang gibt es eine Datei. Dann kommen Korrekturen, Rückmeldungen und neue Ideen. Nach einigen Wochen sieht der Ordner so aus:

```
Satzung.docx
Satzung_Entwurf2.docx
Satzung_Entwurf2_Anmerkungen_Kassenwart.docx
Satzung_final.docx
Satzung_final_neu.docx
Satzung_final_neu_NICHT_LÖSCHEN.docx
```

Wer diesen Ordner nach drei Monaten öffnet, steht vor Rätseln. Welche Datei hat der Vorstand beschlossen? Was hat sich zwischen »final« und »final_neu« geändert? Gab es eine Anmerkung des Kassenwarts, die noch fehlt? Die Antworten stecken irgendwo in den Dateien, aber man muss sie mühsam durch Vergleichen herausfinden. Und sobald zwei Menschen beteiligt sind, die sich Fassungen per E-Mail hin- und herschicken, verdoppelt sich das Durcheinander.

Versionskontrolle löst dieses Problem auf eine erstaunlich einfache Weise: Es gibt nur noch **eine** Datei mit einem festen Namen. Ihre ganze Geschichte steckt unsichtbar dahinter, sauber geordnet und jederzeit abrufbar.



Links das bekannte Durcheinander aus Dateikopien, rechts dieselbe Arbeit mit Versionskontrolle: eine Datei und eine geordnete Geschichte aus festgehaltenen Ständen.

Was eine Versionskontrolle leistet

Die Vorteile lassen sich in fünf Punkten zusammenfassen:

- **Nichts geht verloren.** Jeder festgehaltene Stand bleibt erhalten. Wer heute einen Absatz löscht, kann ihn in einem Jahr noch zurückholen.
- **Änderungen werden nachvollziehbar.** Zu jedem Stand gibt es eine Beschreibung: was geändert wurde und warum.
- **Vergleichen wird einfach.** Das Programm zeigt genau, welche Zeilen hinzugekommen, verändert oder verschwunden sind.
- **Ausprobieren wird gefahrlos.** Man kann eine Idee auf einem Seitenzweig verfolgen, ohne den funktionierenden Stand anzurühren.
- **Zusammenarbeit wird geordnet.** Mehrere Menschen arbeiten an denselben Dateien, und die Versionskontrolle führt ihre Änderungen zusammen.

Der vierte Punkt wird oft unterschätzt. Wer weiß, dass er jederzeit zurückkann, traut sich mehr. Mutige Umbauten, radikale Kürzungen und Experimente werden möglich, weil der Rückweg immer offensteht.

Für wen sich das lohnt

Versionskontrolle wurde für Programmcode erfunden, und dort ist sie heute selbstverständlich. Kaum ein Softwareprojekt der Welt arbeitet ohne. Aber das Prinzip passt überall, wo Text in Dateien entsteht und sich über die Zeit verändert:

Wer	Was versioniert wird	Größter Nutzen
Programmierende	Quellcode, Konfiguration	Fehler finden, gemeinsam arbeiten
technische Redaktion	Handbücher, Anleitungen	Fassungen für Gerätegenerationen
Übersetzerinnen	Ausgangs- und Zieltexte	Änderungen im Original erkennen
Studierende	Abschlussarbeiten, Daten, Auswertungen	nichts verlieren, Stand belegen
Webseitenbetreiber	HTML, CSS, Inhalte	Fehler zurücknehmen
Systembetreuer	Konfigurationsdateien	wissen, wer was umgestellt hat
Vereine, Gremien	Satzungen, Ordnungen, Protokolle	beschlossene Fassungen nachweisen
Musikerinnen	Notensätze in Textformaten	Arrangements vergleichen

Am besten funktioniert Versionskontrolle mit **Textdateien**, also Dateien, die man mit einem einfachen Editor öffnen und lesen kann. Programmcode gehört dazu, aber auch Markdown, HTML, CSV-Tabellen und viele Konfigurationsformate. Word- und Excel-Dateien lassen sich ebenfalls versionieren, der Vergleich ist aber schwieriger. Kapitel 20 widmet sich diesem Thema ausführlich.

> LERNPROJEKT **Drei Ordner und kein Überblick**
>
> Anke Rüther hat den Vertrag mit der Firma Kessler Radtechnik in Kamen unterschrieben. Das Handbuch für die neue Anhängerserie »Kessler Kargo« soll in drei Sprachen erscheinen und jedes Jahr aktualisiert werden. Beim ersten Treffen hat ihr der Produktmanager einen USB-Stick mitgegeben. Darauf liegen 47 Dateien, darunter »Handbuch_Entwurf_TS_final2.docx« und »Handbuch_Entwurf_TS_final2_Korr_MK.docx«. Niemand bei Kessler kann sagen, welche Fassung die aktuelle ist. Anke legt den Stick neben ihre Kaffeetasse und starrt auf die Liste. Am Nachbarschreibtisch dreht sich Dilara Aksoy um. »Darf ich

raten? Final, final2 und final2 mit Korrekturen?« Anke nickt. »Das passiert dir nie wieder«, sagt Dilara. »Wenn du mir einen Abend gibst.«

Lokal und verteilt

Es gibt zwei grundlegende Bauarten von Versionskontrolle. Bei einer **zentralen Versionskontrolle** liegt die gesamte Geschichte auf einem Server. Wer arbeiten will, holt sich den aktuellen Stand, ändert etwas und schickt die Änderung zurück. Ohne Verbindung zum Server kann man keine Stände festhalten und keine alten Fassungen ansehen. Bekannte Vertreter dieser Bauart sind CVS und Subversion, die in den 1990er- und 2000er-Jahren weit verbreitet waren.

Bei einer **verteilten Versionskontrolle** hat jeder Beteiligte eine vollständige Kopie der gesamten Geschichte auf dem eigenen Rechner. Man kann offline arbeiten, Stände festhalten und alte Fassungen vergleichen, ganz ohne Server. Ein Server dient nur noch als gemeinsamer Treffpunkt, an dem man seine Arbeit mit anderen abgleicht. Git ist eine verteilte Versionskontrolle. Das macht es schnell, robust und für Einzelpersonen genauso nützlich wie für große Teams.

Für dich bedeutet das: Du brauchst kein Konto bei irgendeinem Dienst, um mit Git anzufangen. Alles, was du in den ersten vierzehn Kapiteln lernst, funktioniert ausschließlich auf deinem eigenen Rechner.

VERTIEFUNG **Wie Git entstand**

Git wurde 2005 von Linus Torvalds geschrieben, dem Erfinder des Linux-Betriebssystems. Die Linux-Entwickler hatten bis dahin ein kommerzielles Werkzeug namens BitKeeper benutzt, dessen Hersteller die kostenlose Nutzung beendete. Torvalds brauchte schnell einen Ersatz, der mit Tausenden Beteiligten und Millionen Zeilen Code zurechtkam. Nach wenigen Wochen verwaltete Git bereits den Linux-Code. Seit Juli 2005 kümmert sich Junio Hamano um die Weiterentwicklung. Der Name ist ein englisches Schimpfwort für einen sturen, unangenehmen Menschen. Torvalds hat gescherzt, er benenne alle seine Projekte nach sich selbst.

Git, GitHub und andere Namen

Ein häufiges Missverständnis betrifft die Namen. **Git** ist das Programm auf deinem Rechner. Es ist freie Software, gehört nieman-

dem und kostet nichts. **GitHub** ist eine Plattform im Internet, auf der man Git-Projekte speichern, teilen und gemeinsam bearbeiten kann. Sie gehört seit 2018 dem Konzern Microsoft. Daneben gibt es weitere Plattformen wie **GitLab**, **Codeberg** oder **Bitbucket**, und man kann auch einen eigenen Server betreiben, etwa mit **Forgejo**. Alle diese Plattformen benutzen Git, aber Git braucht keine von ihnen.

Diese Trennung ist wichtig für deine Unabhängigkeit. Deine Projektgeschichte liegt vollständig auf deinem Rechner. Du kannst sie jederzeit zu einer anderen Plattform umziehen, ohne etwas zu verlieren. Kapitel 17 vergleicht die Plattformen ausführlich.

Was Versionskontrolle nicht ist

Versionskontrolle ist **keine Datensicherung**. Wenn die Festplatte stirbt und die Projektgeschichte nur dort lag, ist sie verloren, mitsamt allen Versionen. Erst eine Kopie auf einem anderen Gerät oder Server schützt davor. Kapitel 26 zeigt, wie beides zusammenspielt.

Versionskontrolle ist auch **keine automatische Aufzeichnung**. Anders als die Versionsgeschichte mancher Cloud-Dienste speichert Git nicht von selbst jede Minute einen Stand. Du entscheidest, wann ein Stand festgehalten wird, und du beschreibst ihn. Das klingt zunächst nach mehr Arbeit. In der Praxis ist es ein großer Vorteil, weil die Geschichte aus sinnvollen Schritten besteht statt aus Tausenden zufälligen Momentaufnahmen.

Und Versionskontrolle ist **kein Ersatz für Absprachen**. Wenn zwei Menschen denselben Absatz gleichzeitig völlig unterschiedlich umschreiben, kann kein Programm entscheiden, welche Fassung besser ist. Es kann den Konflikt nur deutlich zeigen. Kapitel 12 erklärt, wie man solche Konflikte löst.

Ein kurzer Blick auf andere Systeme

Git ist heute so verbreitet, dass viele Menschen Versionskontrolle und Git für dasselbe halten. Es gibt aber andere Systeme. **Subversion**, kurz SVN, arbeitet zentral und ist in manchen Firmen und Behörden noch im Einsatz. **Mercurial** ist wie Git verteilt und galt lange als einsteigerfreundlicher, hat aber viel Verbreitung verloren. **Jujutsu** ist ein jüngeres Werkzeug, das auf Git aufsetzt und manches anders löst. Kapitel 28 stellt es kurz vor.

Wer Git beherrscht, findet sich in den anderen Systemen schnell zurecht, weil die Grundideen dieselben sind. Umgekehrt gilt das weniger. Für den Einstieg ist Git deshalb die richtige Wahl.

Werkstatt: Deine Dateigeschichte untersuchen

1. Suche auf deinem Rechner einen Ordner, in dem es mehrere Fassungen derselben Datei gibt.
2. Zähle die Kopien. Notiere, wie die Dateinamen gebildet wurden.
3. Versuche herauszufinden, welche Fassung die aktuelle ist. Wie lange brauchst du dafür?
4. Wähle zwei Fassungen aus und versuche, die Unterschiede zu finden.
5. Überlege, welche Informationen dir fehlen: Wer hat geändert? Wann? Warum?
6. Notiere ein Projekt, das du in diesem Buch unter Versionskontrolle stellen möchtest. Es sollte aus mehreren Dateien bestehen und dir wichtig sein.

Häufige Fragen

Muss ich programmieren können, um Git zu benutzen? Nein. Git verwaltet Dateien, ganz gleich, was darin steht. Viele Menschen nutzen es ausschließlich für Texte.

Kostet Git etwas? Nein. Git ist freie Software und darf von jedem kostenlos genutzt, weitergegeben und verändert werden.

Brauche ich ein GitHub-Konto? Nicht, um anzufangen. Ein Konto auf einer Plattform brauchst du erst, wenn du deine Arbeit mit anderen teilen oder auf einem Server sichern möchtest.

Ist Git nicht viel zu kompliziert für mich? Git hat sehr viele Befehle, aber für die tägliche Arbeit reichen etwa zehn. Wer das Grundmodell verstanden hat, kommt erstaunlich weit.

Funktioniert Git auf meinem Betriebssystem? Git läuft auf Windows, macOS und Linux. Kapitel 3 zeigt die Installation für alle drei.

Kann ich Git auch für Fotos verwenden? Grundsätzlich ja, aber Git ist für Textdateien gebaut. Für viele große Bilder gibt es Erweiterungen, die Kapitel 21 vorstellt.

Was passiert mit meinen Dateien, wenn ich Git wieder deinstalliere? Nichts. Deine Dateien bleiben, wie sie sind. Nur die gespeicherte Geschichte ist dann ohne Git nicht mehr bequem lesbar.

Speichert Git meine Daten irgendwo im Internet? Nein, nicht von selbst. Git arbeitet lokal. Erst wenn du ausdrücklich einen

Server einrichtest und deine Arbeit dorthin schickst, verlässt sie deinen Rechner.

> **KURZ GESAGT** **Das Wichtigste aus Kapitel 1**
>
> Eine Versionskontrolle hält festgelegte Stände eines ganzen Ordners fest, mit Datum, Namen und Beschreibung. Sie ersetzt Dateikopien mit wechselnden Namen, macht Änderungen vergleichbar und Experimente gefahrlos. Git ist das verbreitetste System und arbeitet verteilt: Jeder hat die vollständige Geschichte auf dem eigenen Rechner. Plattformen wie GitHub oder Codeberg sind Treffpunkte, nicht Voraussetzung. Versionskontrolle ersetzt weder Datensicherung noch Absprachen.

KAPITEL 2

Die Grundideen

Bevor du den ersten Befehl eintippst, lohnt sich ein Blick auf die Begriffe, mit denen Git arbeitet. Es sind nur eine Handvoll, aber sie tauchen überall auf. Wer sie einmal verstanden hat, liest Anleitungen, Fehlermeldungen und Hilfeseiten mit ganz anderen Augen. Dieses Kapitel erklärt sie ohne Technik, mit Bildern aus dem Alltag.

Das Repository: der Projektordner mit Gedächtnis

Ein **Repository**, oft kurz »Repo« genannt, ist ein Ordner, dessen Geschichte Git verwaltet. Von außen sieht er aus wie jeder andere Ordner: Darin liegen deine Dateien und Unterordner, die du ganz normal öffnen, bearbeiten und speichern kannst. Zusätzlich enthält er einen versteckten Unterordner namens `.git`. Darin bewahrt Git die gesamte Geschichte auf, also alle festgehaltenen Stände, ihre Beschreibungen und einiges Verwaltungswissen.

Diesen versteckten Ordner rührst du nie von Hand an. Er ist Gits Gedächtnis. Wenn du ihn löschst, ist die ganze Geschichte weg, und aus dem Repository wird wieder ein gewöhnlicher Ordner. Deine aktuellen Dateien bleiben dabei unverändert.

Ein Repository ist immer ein Projekt. Für das Handbuch gibt es eines, für das Spiel ein anderes, für die Vereinssatzung ein drittes. Man könnte alles in ein einziges großes Repository packen, aber das macht die Geschichte unübersichtlich. Als Faustregel gilt: Was gemeinsam entsteht und gemeinsam veröffentlicht wird, gehört in ein Repository.

Der Commit: ein festgehaltener Stand

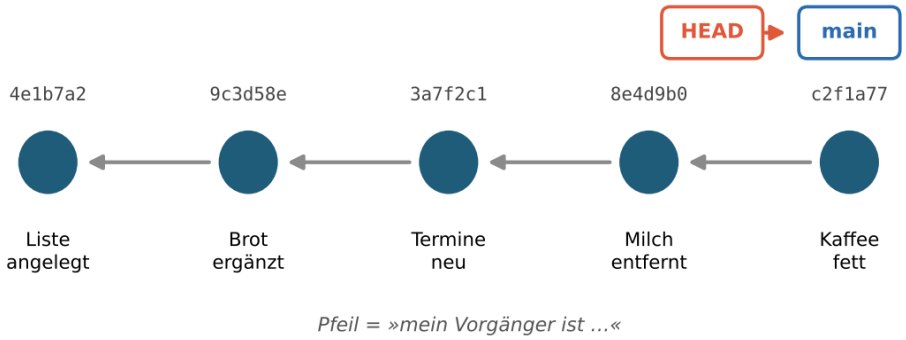
Das Herzstück von Git ist der **Commit**. Ein Commit ist ein festgehaltener Stand des gesamten Projekts zu einem bestimmten Zeitpunkt. Man kann ihn sich wie ein Foto vorstellen: Es zeigt alle Dateien genau so, wie sie in dem Moment waren, als du auf den Auslöser gedrückt hast.

Zu jedem Commit gehören:

- der Inhalt aller Dateien zu diesem Zeitpunkt,

- eine **Commit-Nachricht**, die beschreibt, was sich geändert hat,
- Name und E-Mail-Adresse der Person, die den Commit erstellt hat,
- Datum und Uhrzeit,
- ein Verweis auf den vorherigen Commit,
- eine eindeutige Kennung, die **Commit-ID**.

Der Verweis auf den Vorgänger ist entscheidend. Durch ihn bilden die Commits eine Kette, die vom ersten Stand bis zum heutigen reicht. Diese Kette ist die **Geschichte** des Projekts, im Englischen »history«. Git nennt sie auch den **Verlauf** oder das **Log**.



Commits bilden eine Kette: Jeder festgehaltene Stand verweist auf seinen Vorgänger. HEAD markiert den Stand, an dem du gerade arbeitest.

Das Wort »Commit« kommt aus dem Englischen und bedeutet so viel wie »sich verpflichten« oder »verbindlich festlegen«. Im deutschen Sprachgebrauch hat sich das englische Wort durchgesetzt, man sagt »einen Commit machen« oder »committen«. Ich bleibe in diesem Buch dabei, weil du es in jeder Anleitung so wiederfindest.

Die Commit-ID: ein Fingerabdruck

Jeder Commit hat eine Kennung, die etwa so aussieht:

```
3f9a2c41e8b07d5a1c6e92f0b4d8a17e5c3b9f02
```

Diese lange Folge aus Ziffern und den Buchstaben a bis f ist kein Zufall. Git berechnet sie aus dem Inhalt des Commits: aus allen Dateien, der Nachricht, dem Namen, dem Datum und dem Vorgänger. Ändert sich auch nur ein einziges Zeichen, entsteht eine völlig andere Kennung. Man nennt ein solches Verfahren eine **Prüfsumme** oder einen **Hash**.

Daraus folgen zwei wichtige Eigenschaften. Erstens ist jede Kennung praktisch einmalig, auch über Rechner hinweg. Zweitens kann

man einen Commit nicht unbemerkt nachträglich verändern. Wer das versucht, erzeugt einen neuen Commit mit neuer Kennung, und alle späteren Commits, die auf den alten verweisen, passen nicht mehr dazu. Die Geschichte schützt sich gewissermaßen selbst.

Im Alltag musst du die lange Kennung nie abtippen. Git versteht auch die ersten sieben Zeichen, also `3f9a2c4`, solange sie im Projekt eindeutig sind. Meistens klickst oder kopierst du sie ohnehin.

> **VERTIEFUNG SHA-1 und SHA-256**
>
> Das Verfahren, mit dem Git die Kennungen berechnet, heißt seit 2005 **SHA-1**. Es erzeugt 40 Zeichen lange Prüfsummen. Forschern ist es 2017 gelungen, zwei verschiedene Dateien mit derselben SHA-1-Prüfsumme zu erzeugen. Für Git ist das bisher keine praktische Gefahr, weil Git zusätzliche Schutzmaßnahmen eingebaut hat. Trotzdem arbeitet das Projekt seit Jahren am Umstieg auf **SHA-256** mit 64 Zeichen. Git 3.0 soll neue Repositories standardmäßig mit SHA-256 anlegen. Bestehende Projekte bleiben davon unberührt. Für dich ändert sich nur die Länge der Kennungen.

Der Diff: was sich geändert hat

Ein **Diff** ist die Darstellung der Unterschiede zwischen zwei Ständen. Git zeigt ihn Zeile für Zeile: Zeilen, die neu hinzugekommen sind, werden mit einem Pluszeichen markiert, meist grün. Zeilen, die entfernt wurden, bekommen ein Minuszeichen und sind rot. Eine veränderte Zeile erscheint als entfernte alte und hinzugefügte neue Zeile.

```
- Der Anhänger trägt bis zu 40 kg.
+ Der Anhänger trägt bis zu 45 kg.
```

Diffs sind das wichtigste Werkzeug zum Verstehen einer Geschichte. Statt zwei vollständige Dateien nebeneinanderzulegen und mit dem Finger Zeile für Zeile zu vergleichen, siehst du sofort, was anders ist. Bei Textdateien funktioniert das hervorragend. Bei Word-Dateien sieht Git zunächst nur, dass sich irgendetwas in einer Binärdatei geändert hat. Kapitel 20 zeigt, wie man trotzdem lesbare Vergleiche bekommt.

Der Branch: ein Zweig für eine Idee

Ein **Branch**, auf Deutsch **Zweig**, ist eine eigene Entwicklungslinie innerhalb eines Repositorys. Stell dir die Geschichte als Weg vor. Normalerweise geht er geradeaus: Commit folgt auf Commit. Ein Branch ist eine Abzweigung. Auf ihr kannst du eine Idee ausprobie-

ren, eine größere Umarbeitung beginnen oder eine neue Sprachfassung aufbauen, während der Hauptweg unberührt bleibt. Wenn die Idee gut war, führst du den Zweig später wieder mit dem Hauptweg zusammen. Wenn nicht, lässt du ihn einfach liegen.

Jedes Repository hat mindestens einen Branch, den Hauptzweig. Er heißt in neuen Projekten meist `main`, in älteren oft `master`. Kapitel 10 erklärt Branches ausführlich. Hier reicht es zu wissen, dass es sie gibt und dass sie in Git sehr leicht und schnell zu erstellen sind.

HEAD: Hier stehst du

Bei so vielen Ständen und Zweigen braucht Git eine Markierung für den Ort, an dem du gerade arbeitest. Diese Markierung heißt **HEAD**. Meistens zeigt HEAD auf den neuesten Commit des Zweigs, auf dem du arbeitest. Wenn du einen neuen Commit erstellst, rückt HEAD mit nach vorn. Wenn du zu einem anderen Zweig wechselst, springt HEAD dorthin.

Du wirst den Namen HEAD in vielen Befehlen und Meldungen sehen. Lies ihn einfach als »der Stand, an dem ich gerade bin«.

Die wichtigsten Begriffe im Überblick

Begriff	Bedeutung	Bild aus dem Alltag
Repository	Projektordner mit Geschichte	Aktenordner mit Archiv
Commit	festgehaltener Stand	Foto des ganzen Schreibtischs
Commit-Nachricht	Beschreibung des Stands	Beschriftung auf der Rückseite
Commit-ID	eindeutige Kennung	Fingerabdruck
Diff	Unterschied zwischen Ständen	Korrekturfahne
Branch	eigene Entwicklungslinie	Abzweigung vom Weg
HEAD	aktueller Arbeitsstand	Lesezeichen
Merge	Zusammenführen zweier Zweige	Wege, die sich wieder treffen
Remote	entfernte Kopie auf einem Server	Zweitschlüssel bei den Nachbarn

LERNPROJEKT Dilaras Serviette

Am Abend sitzen Anke und Dilara im leeren Büro. Dilara zieht eine Papierserviette aus der Schublade und malt Kreise darauf, durch Pfeile verbunden. »Jeder Kreis ist ein Stand deines Handbuchs. Du sagst, wann ein Foto gemacht wird, und schreibst drauf, was passiert ist.« Anke runzelt die Stirn. »Und wenn ich am Mittwoch etwas lösche, was ich am Freitag wieder brauche?« Dilara tippt auf einen Kreis in der Mitte. »Dann holst du es von hier.« Sie malt eine Abzweigung. »Und hier fängst du die niederländische Fassung an, ohne die deutsche anzurühren.« Anke nimmt die Serviette, faltet sie und steckt sie in ihr Notizbuch. Die USB-Stick-Dateien wird sie trotzdem einzeln durchgehen müssen. Aber zum ersten Mal weiß sie, wohin sie die Ordnung bringen will.

Warum Git nicht jede Sekunde speichert

Viele Menschen fragen sich am Anfang, warum sie Commits selbst auslösen müssen. Ein Programm könnte doch einfach jede gespeicherte Änderung festhalten. Die Antwort liegt in der Qualität der Geschichte. Eine Aufzeichnung jedes Tastendrucks wäre zwar vollständig, aber nutzlos. Niemand möchte sich durch Tausende Zwischenstände wühlen, um den Moment zu finden, in dem eine Idee fertig war.

Ein Commit markiert dagegen einen sinnvollen Schritt: »Kapitel Sicherheitshinweise übersetzt«, »Bremsdiagramm ersetzt«, »Tippfehler in der Einleitung korrigiert«. Diese Schritte lassen sich lesen, verstehen und bei Bedarf einzeln zurücknehmen. Die Disziplin, die Git verlangt, zahlt sich jedes Mal aus, wenn man die Geschichte später durchsucht. Kapitel 6 zeigt, wie gute Commits aussehen.

Wie Git den Inhalt speichert

Es wäre eine verständliche Vermutung, dass Git bei jedem Commit nur die Unterschiede zum vorherigen Stand speichert. Tatsächlich denkt Git anders: Jeder Commit ist ein vollständiges Abbild aller Dateien. Damit das nicht unnötig Platz verschwendet, legt Git jeden Dateiinhalt nur ein einziges Mal ab. Wenn sich eine Datei zwischen zwei Commits nicht ändert, verweist der neue Commit einfach auf denselben gespeicherten Inhalt. Zusätzlich packt Git ältere Daten in regelmäßigen Abständen platzsparend zusammen.

Für dich hat das einen angenehmen Nebeneffekt: Ein Repository mit hundert Commits eines Textprojekts ist oft kaum größer als die aktuellen Dateien selbst. Und das Wiederherstellen eines alten