Vorwort

Warum Git?

Git hat eine rasante Erfolgsgeschichte hinter sich. Im April 2005 begann Linus Torvalds, Git zu implementieren, weil er keinen Gefallen an den damals verfügbaren Open-Source-Versionsverwaltungen fand. Heute, im Frühjahr 2019, liefert Google Millionen von Suchtreffern, wenn man nach »git version control« sucht. Für neue Open-Source-Projekte ist es zum Standard geworden, und viele große Open-Source-Projekte sind zu Git migriert.

Arbeiten mit Branches: Wenn viele Entwickler gemeinsam an einer Software arbeiten, entstehen parallele Entwicklungsstränge, die immer wieder auseinanderlaufen und zusammengeführt werden müssen. Genau dafür ist Git entwickelt worden. Es bietet daher umfassende Unterstützung zum *Branchen*, *Mergen*, *Rebasen* und *Cherry-Picken*.

Flexibilität in den Workflows: Manche sagen, dass Git im Grunde gar keine Versionsverwaltung sei, sondern ein Baukasten, aus dem sich jeder seine eigene Versionsverwaltung zusammensetzen kann. Git ist außergewöhnlich flexibel. Ein einzelner Entwickler kann es für sich allein nutzen, agile Teams finden leichtgewichtige Arbeitsweisen damit, aber auch große internationale Projekte mit zahlreichen Entwicklern an mehreren Standorten können passende Workflows entwickeln.

Contribution: Die meisten Open-Source-Projekte existieren durch freiwillige Beiträge von Entwicklern. Es ist wichtig, das Beitragen so einfach wie nur möglich zu machen. Bei zentralen Versionsverwaltungen wird dies oft erschwert, weil man nicht jedem schreibenden Zugriff auf das Repository geben möchte. Jeder kann ein Git-Repository klonen, damit vollwertig arbeiten und dann später die Änderungen weitergeben: »Mit Forks entwickeln« (Seite 173).

Nachvollziehbare Herkunft von Sourcecode: Die Entwickler von Git haben es als *Content Tracker* bezeichnet. Damit meinen sie ein Werkzeug, das die Herkunft von Inhalten, insbesondere Source-

code, aufzeigen kann. Git kann dies selbst dann, wenn Code zusammengeführt wurde (*Merge*) oder Dateien verschoben und umbenannt wurden. Sogar kopierte Codeabschnitte können erkannt und zugeordnet werden.

Performance: Auch bei Projekten mit vielen Dateien und langen Historien bleibt Git schnell. In weniger als einer halben Minute wechselt es zum Beispiel von der aktuellen Version auf eine sechs Jahre ältere Version der Linux-Kernel-Sourcen – auf einem kleinen MacBook Air. Das kann sich sehen lassen, wenn man bedenkt, dass über 200.000 Commits und 40.000 veränderte Dateien dazwischenliegen.

Robust gegen Fehler und Angriffe: Da die Historie auf viele dezentrale Repositorys verteilt wird, ist ein schwerwiegender Datenverlust unwahrscheinlich. Eine genial simple Datenstruktur im Repository sorgt dafür, dass die Daten auch in ferner Zukunft interpretierbar bleiben. Der durchgängige Einsatz kryptografischer Prüfsummen erschwert es Angreifern, Repositorys unbemerkt zu korrumpieren.

Offline- und Multisite-Entwicklung: Die dezentrale Architektur macht es leicht, offline zu entwickeln, etwa unterwegs mit dem Laptop. Bei der Entwicklung an mehreren Standorten ist weder ein zentraler Server noch eine dauerhafte Netzwerkverbindung erforderlich.

Administrierbarkeit: Git ist einfach zu betreiben und zu administrieren. Alle Daten und Konfigurationen werden in einfachen Dateien gespeichert. Für Backups oder Umzüge genügen die Standardtools des Betriebssystems. Es muss kein Git-spezifischer Dienst eingerichtet werden. Alle Operationen werden durch Kommandozeilenbefehle bereitgestellt. Repositorys werden meist über SSH oder HTTP zugänglich gemacht, sodass man die Authentifizierung und Autorisierung des Betriebssystems bzw. Webservers nutzen kann. Zahlreiche mächtige Befehle erlauben es, das Repository zu manipulieren. Die dezentrale Natur von Git macht es leicht, Änderungen zuerst an einem Klon zu erproben, bevor man sie öffentlich macht.

Starke Open-Source-Community: Neben der detaillierten offiziellen Dokumentation unterstützen zahlreiche Anleitungen, Foren, Wikis etc. den Anwender. Es existiert ein Ökosystem aus Tools, Hosting-Plattformen, Publikationen, Dienstleistern und Plugins für Entwicklungsumgebungen, und es wächst stark.

Erweiterbarkeit: Git bietet neben komfortablen Befehlen für den Anwender auch elementare Befehle, die einen direkteren Zugang zum Repository erlauben. Dies macht Git sehr flexibel und ermöglicht individuelle Anwendungen, die über das hinausgehen, was Git von Haus aus bietet.

Neues in der fünften Auflage

Git ist nun schon seit vielen Jahren im Einsatz. Die Projekte werden immer größer, die Entwickler werden mehr, und daraus ergeben sich neue Herausforderungen. Deshalb haben wir dem Umgang mit sehr großen Repositorys ein eigenes Kapitel gewidmet, in dem Konzepte wie Sparse Checkout und Shallow Clone sowie Tools wie Watchman vorgestellt werden.

Große Repositorys *→ Seite 285*

Ein Abschnitt über den im Alltag sehr nützlichen `show`-Befehl, den wir bisher übersehen hatten, ist jetzt mit an Bord.

Das Konzept des HEAD-Commits, welches bisher in verstreuten Anmerkungen nur angerissen wurde, hat nun einen eigenen Abschnitt erhalten. Ebenso erläutern wir jetzt in einem Abschnitt gebündelt die wichtigsten Schreibweisen für Commit-Parameter.

Das HEAD-Commit *→ Seite 35*
Schreibweisen für Commits *→ Seite 40*

Von Lesern und Seminarteilnehmern gab es immer mal wieder Fragen zur Bedeutung des Master-Branch. Deshalb haben wir einen Abschnitt hierzu ergänzt.

***Der* Master-Branch** *→ Seite 68*

Auch für das `origin`-Repository haben wir einen erläuternden Abschnitt hinzugefügt.

***Das* Origin-Repository** *→ Seite 99*

Dringend nötig war auch die Überarbeitung des Kapitels über Jenkins, da sich sowohl beim Build-Server als auch bei den Hosting-Plattformen für Git in den letzten Jahren einiges bewegt hat.

Integration mit Jenkins *→ Seite 273*

Dank des Feedbacks von Lesern konnten wir darüber hinaus in den Grundlagen-Kapiteln (1 bis 12) viele Dinge präziser und hoffentlich auch verständlicher formulieren.

Feedback zum Buch

Falls Sie Fehler entdecken, Fragen oder Vorschläge haben, können Sie Feedback über unsere Website geben:

http://kapitel26.github.io/feedback

Oder Sie schreiben uns eine E-Mail:

`git@eToSquare.de`

Ein Buch für professionelle Entwickler

Wenn Sie Entwickler sind, im Team Software herstellen und wissen wollen, wie man Git effektiv einsetzt, dann halten Sie jetzt das richtige Buch in der Hand. Dies ist kein theorielastiger Wälzer und auch kein umfassendes Nachschlagewerk. Es beschreibt nicht alle Befehle von Git (es sind mehr als 100). Es beschreibt erst recht nicht alle Optionen (einige Befehle bieten über 50 an). Stattdessen beschreibt dieses Buch, wie man Git in typischen Projektsituationen einsetzen kann, z. B. wie man ein Git-Projekt aufsetzt oder wie man mit Git ein Release durchführt.

Ein Projekt aufsetzen → *Seite 133*
Release durchführen → *Seite 195*

Die Zutaten

Gleich ausprobieren! → *Seite 9*
Was sind Commits? → *Seite 31*
Tipps und Tricks → *Seite 117*
Workflow-Verzeichnis → *Seite 325*

Erste Schritte: Auf weniger als einem Dutzend Seiten zeigt ein Beispiel alle wichtigen Git-Befehle.

Einführung: Auf weniger als hundert Seiten erfahren Sie, was man benötigt, um mit Git im Team arbeiten zu können. Zahlreiche Beispiele zeigen, wie man die wichtigsten Git-Befehle anwendet. Darüber hinaus werden wesentliche Grundbegriffe, wie zum Beispiel Commit, Repository, Branch, Merge oder Rebase, erklärt, die Ihnen helfen zu verstehen, wie Git funktioniert, damit Sie die Befehle gezielter einsetzen können. Hier finden Sie auch einen Abschnitt mit Tipps und Tricks, die man nicht jeden Tag braucht, die aber manchmal nützlich sein können.

Workflows: Workflows beschreiben Szenarien, wie man Git im Projekt einsetzen kann, zum Beispiel wenn man ein Release durchführen möchte: »Periodisch Releases durchführen« (Seite 195). Für jeden Workflow wird beschrieben,

- welches Problem er löst,
- welche Voraussetzungen dazu gegeben sein müssen und
- wer wann was zu tun hat, damit das gewünschte Ergebnis erreicht wird.

»**Warum nicht anders?**«**-Abschnitte:** Jeder Workflow beschreibt genau einen konkreten Lösungsweg. In Git gibt es häufig sehr unterschiedliche Wege, um dasselbe Ziel zu erreichen. Im letzten Teil eines jeden Workflow-Kapitels wird erklärt, warum wir genau diese eine Lösung gewählt haben. Dort werden auch Varianten und Alternativen erwähnt, die für Sie interessant sind, wenn in Ihrem Projekt andere Voraussetzungen gegeben sind oder wenn Sie mehr über die Hintergründe wissen wollen.

Schritt-für-Schritt-Anleitungen: Häufig benötigte Befehlsfolgen, wie zum Beispiel »Branch erstellen« (Seite 66), haben wir in Schritt-für-Schritt-Anleitungen beschrieben.

Schritt-für-Schritt-Anleitungen
→ Seite 323

Warum Workflows?

Git ist extrem flexibel. Das ist gut, weil es für die unterschiedlichsten Projekte taugt. Vom einzelnen Sysadmin, der »mal eben« ein paar Shell-Skripte versioniert, bis hin zum Linux-Kernel-Projekt, an dem Hunderte von Entwicklern arbeiten, ist jeder damit gut bedient. Diese Flexibilität hat jedoch ihren Preis. Wer mit Git zu arbeiten beginnt, muss viele Entscheidungen treffen. Zum Beispiel:

- In Git hat man dezentrale Repositorys. Aber möchte man wirklich nur dezentral arbeiten? Oder richtet man doch lieber ein zentrales Repository ein?
- Git unterstützt zwei Richtungen für den Datentransfer: Push und Pull. Benutzt man beide? Falls ja: Wofür verwendet man das eine? Wofür das andere?
- Branching und Merging ist eine Stärke von Git. Aber wie viele Branches öffnet man? Einen für jedes Feature? Einen für jedes Release? Oder überhaupt nur einen?

Um den Einstieg zu erleichtern, haben wir 12 Workflows beschrieben:

- Die Workflows sind Arbeitsabläufe für den Projektalltag.
- Die Workflows geben konkrete Handlungsanweisungen.
- Die Workflows zeigen die benötigten Befehle und Optionen.
- Die Workflows eignen sich gut für eng zusammenarbeitende Teams, wie man sie in modernen Softwareprojekten häufig antrifft.
- Die Workflows sind *nicht* die einzige richtige Lösung für das jeweilige Problem. Aber sie sind ein guter Startpunkt, von dem man ausgehen kann, um optimale Workflows für das eigene Projekt zu entwickeln.

Wir konzentrieren uns auf die agile Entwicklung im Team für kommerzielle Projekte, weil wir glauben, dass sehr viele professionelle Entwickler (die Autoren inklusive) in solchen Umgebungen arbeiten. Nicht berücksichtigt haben wir die besonderen Belange der Open-Source-Entwicklung, obwohl es auch dafür sehr interessante Workflows mit Git gibt.

Tipps zum Querlesen

Als Autoren wünschen wir uns natürlich, dass Sie unser Buch von Seite 1 bis Seite 320 am Stück verschlingen, ohne es zwischendrin aus der Hand zu legen. Aber mal ehrlich: Haben Sie genug Zeit, um heute noch mehr als ein paar Seiten zu lesen? Wir vermuten, dass in Ihrem Projekt gerade die Hölle los ist und dass das Arbeiten mit Git nur eines von hundert Themen ist, mit denen Sie sich momentan beschäftigen. Deshalb haben wir uns Mühe gegeben, das Buch so zu gestalten, dass man es gut querlesen kann. Hier sind ein paar Tipps dazu:

Muss ich die Einführungskapitel lesen, um die Workflows zu verstehen?

Falls Sie noch keine Vorkenntnisse in Git haben, sollten Sie das tun. Grundlegende Befehle und Prinzipien sollten Sie kennen, um die Workflows korrekt einsetzen zu können.

Ich habe schon mit Git gearbeitet. Welche Kapitel kann ich überspringen?

Zusammenfassung am Ende der Einführungskapitel

Auf der letzten Seite in jedem Einführungskapitel 1 bis 14 gibt es eine Zusammenfassung der Inhalte in Stichworten. Dort können Sie sehr schnell sehen, ob es in dem Kapitel für Sie noch Dinge zu entdecken gibt oder ob Sie es überspringen können. Die folgenden Kapitel können Sie relativ gut überspringen, weil sie nur für einige Workflows relevant sind:

Überspringen Sie diese Kapitel, wenn Sie es eilig haben.

Kapitel 6, Das Repository
Kapitel 9, Mit Rebasing die Historie glätten
Kapitel 12, Versionen markieren
Kapitel 30, Abhängigkeiten zwischen Repositorys
Kapitel 13, Tipps und Tricks

Wo finde ich was?

Workflow-Verzeichnis *→ Seite 325*

Workflows: Ein Verzeichnis aller Workflows mit Kurzbeschreibungen und Überblicksabbildung finden Sie im Anhang.

Anleitungsverzeichnis *→ Seite 323*

Schritt-für-Schritt-Anleitungen: Wir haben alle Anleitungen im Anhang aufgelistet.

Index *→ Seite 330*

Befehle und Optionen: Wenn Sie beispielsweise wissen wollen, wie man die Option `find-copies-harder` verwendet und zu welchem Befehl sie gehört, dann schauen Sie in den Index. Dort sind fast alle Verwendungen von Befehlen und Optionen aufgeführt. Oft haben wir

die Nummer jener Seite **fett** hervorgehoben, wo Sie am meisten Informationen zu dem Befehl oder der Option finden.

Fachbegriffe: Fachbegriffe, wie zum Beispiel »First-Parent-History« oder »Remote-Tracking-Branch«, finden Sie natürlich auch im Index.

Index → Seite 330

Beispiele und Notation

In den »Erste Schritte«-Kapiteln verwenden wir einige Git-Fachbegriffe, die wir erst in späteren Kapiteln ausführlicher behandeln. Diese sind kursiv gesetzt, z. B. *Repository*. Im Index sind jene Seitenzahlen fett hervorgehoben, wo der Begriff dann definiert oder ausführlicher erläutert wird.

Index → Seite 330

Viele Beispiele in diesem Buch beschreiben wir mit Kommandozeilenaufrufen. Das soll nicht heißen, dass es dafür keine grafischen Benutzeroberflächen gibt. Im Gegenteil: Git bringt zwei einfache grafische Anwendungen bereits mit: `gitk` und `git-gui`. Darüber hinaus gibt es zahlreiche Git-Frontends (z. B. Atlassian SourceTree[1], TortoiseGit[2], SmartGit[3], GitX[4], Git Extensions[5], tig[6], qgit[7]), einige Entwicklungsumgebungen, die Git von Haus aus unterstützen (IntelliJ[8], Xcode 4[9]), und viele Plugins für Entwicklungsumgebungen (z. B. EGit für Eclipse[10], NBGit für NetBeans[11], Git Extensions für Visual Studio[12]). Wir haben uns trotzdem für die Kommandozeilenbeispiele entschieden, weil

***Grafische Werkzeuge für Git** → Seite 310*

- Git-Kommandozeilenbefehle auf allen Plattformen fast gleich funktionieren,
- die Beispiele auch mit künftigen Versionen funktionieren werden,
- man damit Workflows sehr kompakt darstellen kann und weil
- wir glauben, dass das Arbeiten mit der Kommandozeile für viele Anwendungsfälle unschlagbar effizient ist.

[1] *http://www.sourcetreeapp.com/*
[2] *http://code.google.com/p/tortoisegit/*
[3] *http://www.syntevo.com/smartgit/*
[4] *http://gitx.frim.nl/*
[5] *http://gitextensions.github.io/*
[6] *https://jonas.github.io/tig/*
[7] *http://sourceforge.net/projects/qgit/*
[8] *http://www.jetbrains.com/idea/*
[9] *https://developer.apple.com/xcode/*
[10] *http://eclipse.org/egit/*
[11] *https://netbeans.org/kb/docs/ide/git.html*
[12] *http://gitextensions.github.io/*

In den Beispielen arbeiten wir mit der Bash-Shell, die auf Linux- und Mac-OS-Systemen standardmäßig vorhanden ist. Auf Windows-Systemen kann man die »Git-Bash«-Shell (sie ist in der »msysgit«-Installation enthalten) oder »cygwin« verwenden. Die Kommandozeilenaufrufe stellen wir wie folgt dar:

```
> git commit
```

An den Stellen, wo es inhaltlich interessant ist, zeigen wir auch die Antwort, die Git geliefert hat, in etwas kleinerer Schrift dahinter an:

```
> git --version
  git version 2.3.7
```

Eine kurze Einführung in das Arbeiten mit Git-User-Interfaces zeigen wir am Beispiel von SourceTree in Kapitel »Erste Schritte mit SourceTree« ab Seite 23. Wo wir auf Menüpunkte oder Buttons Bezug nehmen, setzen wir diese in Kapitälchen, z. B. Ansicht | Aktualisieren.

Danksagungen

Danksagungen zur ersten Auflage

Rückblickend sind wir erstaunt, wie viele Leute auf die eine oder andere Weise zum Entstehen dieses Buchs beigetragen haben. Wir möchten uns ganz herzlich bei all jenen bedanken, ohne die dieses Buch nicht das geworden wäre, was es jetzt ist.

An erster Stelle danken wir Anke, Jan, Elke und Annika, die sich inzwischen kaum noch daran erinnern, wie wir ohne einen Laptop unter den Fingern aussehen.

Dann danken wir dem freundlichen Team vom dpunkt.verlag, insbesondere Vanessa Wittmer, Nadine Thiele und Ursula Zimpfer. Besonderer Dank gebührt aber René Schönfeldt, der das Projekt angestoßen und vom ersten Tag bis zur letzten Korrektur begleitet hat. Außerdem bekanken wir uns bei Maurice Kowalski, Jochen Schlosser, Oliver Zeigermann, Ralf Degner, Michael Schulze-Ruhfus und einem halben Dutzend anonymer Gutachter für die wertvollen inhaltlichen Beiträge, die sehr geholfen haben, das Buch besser zu machen. Für den allerersten Anstoß danken wir Matthias Veit, der eines Tages zu Bjørn kam und meinte, Subversion wäre nun doch schon etwas in die Jahre gekommen und man solle sich doch mal nach etwas Schönerem umsehen, zum Beispiel gäbe es da so ein Tool, das die Entwickler des Linux-Kernels neuerdings nutzen würden …

Danksagungen zur zweiten Auflage

Ein besonders herzlicher Dank geht an unseren Leser Herrn Ulrich Windl, der das Buch aufmerksamer gelesen hat als die meisten und uns eine lange Liste von Fragen, Korrektur- und Verbesserungsvorschlägen zugesandt hat. Sie haben wesentlich dazu beigetragen, dass diese Auflage besser und präziser ist als die erste.

Ebenfalls danken wir Henrik Heine, Malte Finsterwalder und Tjabo Vierbücher für gute Hinweise und Korrekturvorschläge.

Danksagungen zur dritten Auflage

Dieses Mal gab es nicht so viel Feedback. Ein paar kleinere Fehlermeldungen haben wir dennoch erhalten, und so konnten wir das Buch wieder ein wenig verbessern. Vielen Dank dafür.

Danksagungen zur vierten Auflage

Nützliche Hinweise haben wir erhalten von: Thomas Braun, Herbert Feichtinger, Jason Stäuble (@shadyhh), Udo Heyn (@UdoHeyn) und Herrn Böckler. Vielen Dank dafür!

Danksagungen zur fünften Auflage

Ein ganz besonderer Dank geht an Dr. Oliver Thilmann, der uns eine lange E-Mail mit sehr vielen Anmerkungen und Verbesserungsvorschlägen geschrieben hat, die uns geholfen hat, diese Auflage präziser und verständlicher zu gestalten. Nützliche Hinweise haben wir außerdem erhalten von: Gerald Schroll (@gschroll), @TheDet und Hampa Brügger (@hampa-git).

»Standing on the Shoulders of Giants«

Ein besonderer Dank geht an Linus Torvalds, Junio C. Hamano und die vielen Committer im Git-Projekt dafür, dass sie der Entwickler-Community dieses fantastische Tool geschenkt haben.