Git

Dezentrale Versionsverwaltung im Team - Grundlagen und Workflows

von
René Preißel, Bjørn Stachmann

2., aktualisierte und erweiterte Auflage

Thematische Gliederung:

Software Engineering

dpunkt.verlag 2013

Verlag C.H. Beck im Internet:
www.beck.de
ISBN 978 3 86490 130 0

Inhaltsverzeichnis: Git – Preißel / Stachmann

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 Juli 2013, liefert Google Millionen von Suchtreffern, wenn man nach »git version control« sucht. Für neue Open-Source-Projekte ist es fast schon zum Standard geworden. Viele große Open-Source-Projekte sind bereits zu Git migriert oder sind dabei, dies zu tun.

Arbeiten mit Branches: Mit Git können viele Entwickler parallel auf dezentralen Repositorys arbeiten. Dabei entstehen viele unterschiedliche Entwicklungsstränge. Die Stärke von Git liegt in den Werkzeugen, die helfen, diese Entwicklungsstränge wieder zusammenzuführen: *Merging*, *Rebasing*, *Cherry-Picking* etc.

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 alleine 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 (siehe auch »Pull-Request« auf Seite 125).

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 Mac-

Book Air. Das kann sich sehen lassen, wenn man bedenkt, dass über 200.000 Commits und 40.000 veränderte Dateien dazwischen liegen.

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.

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 Plug-ins 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.

Zur zweiten Auflage

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. Dieses Buch 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 111*
Release durchführen → *Seite 179*

Die Zutaten

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

Gleich ausprobieren!
→ Seite 9

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.

Was sind Commits?
→ Seite 19

Tipps und Tricks
→ Seite 101

Workflows: Workflows beschreiben Szenarien, wie man Git im Projekt einsetzen kann, zum Beispiel wenn man ein Release durchführen möchte (»Ein Release durchführen« (Seite 179)). Für jeden Workflow wird beschrieben,

- welches Problem er löst,
- welche Voraussetzungen dazu gegeben sein müsssen und
- wer wann was zu tun hat,

damit das gewünschte Ergebnis erreicht wird.

Workflow-Verzeichnis
→ Seite 263

»**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 »Einen Branch verschieben« (Seite 69), haben wir in »Schritt für Schritt«-Anleitungen beschrieben.

»Schritt für Schritt«-Anleitungen
→ Seite 260

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 alles machbar. 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 11 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, so 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 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 speziellen Anforderungen, die sich für Großprojekte ergeben, weil sie die Workflows deutlich aufgebläht hätten und weil wir glauben, dass sie für die meisten Entwickler nicht so interessant sind. Ebenfalls unberücksichtigt bleibt die 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 258 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 gerade 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, lautet die Antwort: am besten ja. 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?

Auf der letzten Seite in jedem Einführungskapitel 1 bis 11 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:

Zusammenfassung am Ende der Einführungskapitel

Kapitel 5, Das Repository
Kapitel 8, Mit Rebasing die Historie glätten
Kapitel 10, Versionen markieren
Kapitel 11, Abhängigkeiten zwischen Repositorys

Überspringen Sie diese Kapitel, wenn Sie es eilig haben.

Wo finde ich was?

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

Workflow-Verzeichnis *→ Seite 263*

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

Anleitungsverzeichnis *→ Seite 260*

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 jene Seitenzahl **fett** hervorgehoben, wo Sie am meisten Informationen zu dem Befehl oder der Option finden.

Index *→ Seite 269*

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

Index *→ Seite 269*

Beispiele und Notation

Grafische Werkzeuge für Git → *Seite 246*

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 Plug-ins 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

- 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.

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 1.8.3.4
```

[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://code.google.com/p/gitextensions/
[6] http://jonas.nitro.dk/tig/
[7] http://sourceforge.net/projects/qgit/
[8] http://www.jetbrains.com/idea/
[9] http://developer.apple.com/technologies/tools/
[10] http://eclipse.org/egit/
[11] http://nbgit.org/
[12] http://code.google.com/p/gitextensions/

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-Kernel 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 Korrekturvorschlägen, Fragen 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.

»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.