

Sujeevan Vijayakumaran

Git

Schnelleinstieg

Versionsverwaltung lernen in 14 Tagen

Einfach und ohne Vorkenntnisse

Zahlreiche
Praxisbeispiele



mitp

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

4

Branches per Rebase zusammenführen und das Aufräumen des Repositorys

4.1 Branches per Rebase zusammenführen 75

4.2 Aufräumen mit Stash und Clean 82

5

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

6

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

7

CI/CD: Continuous Integration und Continuous Delivery

7.1 Der Workflow 154

7.2 GitHub Actions 156

7.3 GitLab CI/CD 159

8

Workflow für Einzelpersonen

8.1 Interaktives Rebasing 166

8.2 Workflow mit einem Branch und Repository für eine Person 179

9

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 Repositories	195
9.5	GitHub-Flow	196
9.6	GitLab-Flow	197
9.7	Weitere Aspekte in Workflows	199

10

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

11

Nachvollziehbare Git-Historien

11.1	Gut dosierte Commits	214
11.2	Gute Commit-Messages	215

12

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 Repositories arbeiten, sei es auf GitHub oder GitLab. Wichtig und für da allgemeine 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 Fragen und 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

1

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 Autorennamen, 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

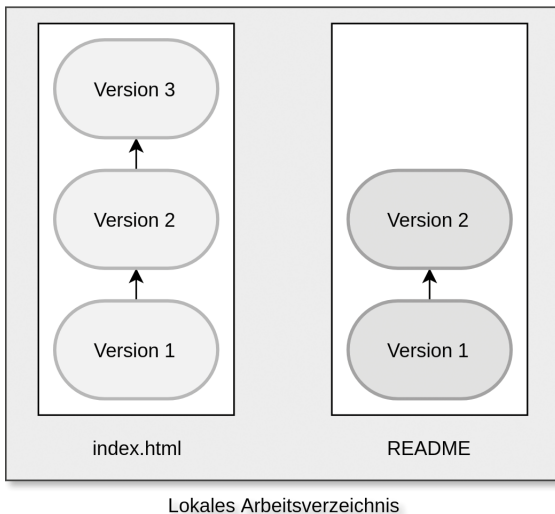


Abb. 1.2: Lokale Versionsverwaltung arbeitet dateibasiert und lediglich lokal.

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

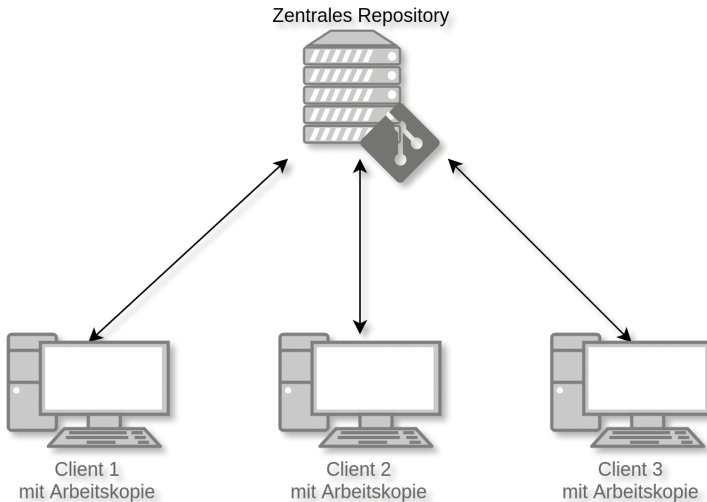


Abb. 1.3: Zentrale Versionsverwaltung arbeitet mit Arbeitskopien auf Clients.

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 Repositories handelt.

1.1.3 Verteilte Versionsverwaltung

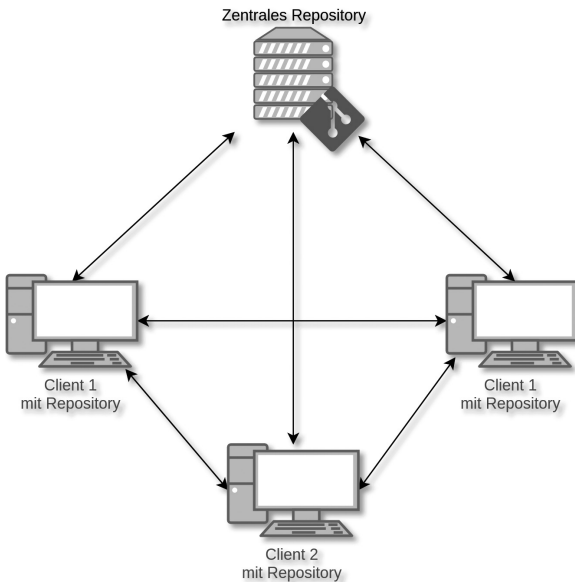


Abb. 1.4: Verteilte Versionsverwaltung arbeitet mit Repositories auf Clients und Servern.

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 Repositories 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 Repositories 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.

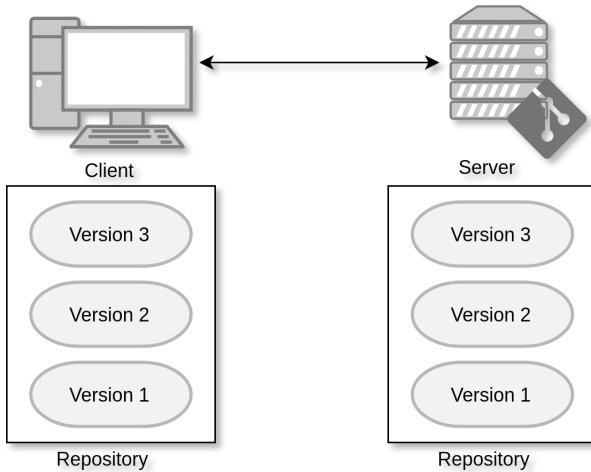


Abb. 1.5: Die Versionshistorie liegt sowohl lokal auf dem Client als auch auf dem Server.

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 loslegen, 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.

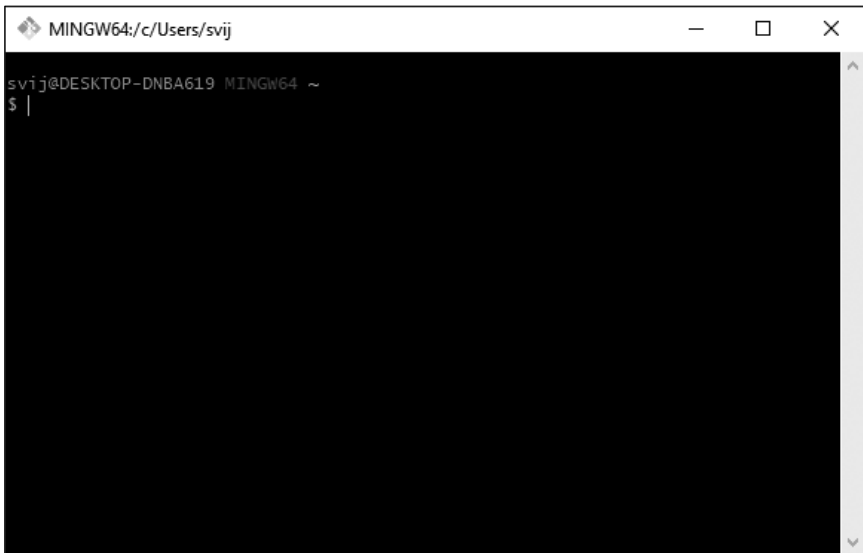


Abb. 1.6: Git-Bash unter Windows

Stichwortverzeichnis

Symbole

--follow	42
.gitconfig	24
.gitignore	87
.gitmodules	233
git-svn	206
--global	24

A

Alias	225
Änderung	
austragen	40
Änderungshistorie	13
Archiv	17
Auto DevOps	163
autosquash	177

B

Bazaar	16, 203
Betriebssystem	18
Binärdatei	
in Git-Repositories	205
Blame	229
blob-Objekt	50
branch	57
Branch	55
Entwicklungsbranch	55
Feature	187
Hotfix	191
löschen	67
Master	56
mergen	64
Release	189
Topic	57
Branch-Management	101
Bugfix-Branch	57

C

cggit	152
Check-Liste	138
checkout	57
Cherry-Picken	195
Code-Analyse	145

Code-Review	134
Commit	
ändern	168
aufteilen	178
entfernen	178
ergänzen	173
initialer	25
Merge	69
Referenzierung	46
Reihenfolge anpassen	173
squashen	175
zusammenführen	175
Commit-Nachricht	41
Community	
Umgang	136
Continuous Integration	145
CONTRIBUTING.md	136
curl	25
CVS	15

D

Datei	
aus Commit ausschließen	30
ignorieren	87
unbeobachtete	27
versionierte	42
detached HEAD	56

E

Entwickler	
E-Mail-Adresse	24
Name	24
Entwicklungsbranch	55

F

Farbausgabe	38
Fast-Forward-Merge	67
Feature-Branch	, 57
Best Practices	188
Fehlersuche	230
Force-Push	142
Forken	129

G

Gerrit	152
Gist	119
Git	
Arbeitsweise	47
git add	28
git bisect	230
git branch -d	67
git branch --no-merged	101
git cat-file	50
git clean	85, 86
git clone	128
git commit --amend	168
git diff	38
git diff HEAD	39
git fetch	95
git fetch --all	109
Git Flow	185
Workflow	192
git help	52
Git-Hilfe	52
GitHub	, 127, 118
Dienste	145
Lizenz	121
Organisationen	119
Registrierung	119
Repository anlegen	119
Repository klonen	128
Repository konfigurieren	126
SSH-Keys	122
GitHub Flavored Markdown	137
GitHub-Issues	136
GitHub-Organisation	144
GitHub-Workflow	129
Git-Identität	24
GitLab	, 115, 146
.gitlab-ci.yml	159
Gruppen	147
Installation	146
Issue-Tracker	150
Jobs	159
Pipeline	159
stages	160
Systemvoraussetzungen	118
GitLab CI/CD	159
script	160

GitLab.com	117
GitLab Enterprise	117
GitLab-Runner	162
git log	34
git log -p	72
git merge	65
git merge --no-ff	188
git mv	42
git pull	96
git pull --rebase	185
git push	99
git push --delete	103
git rebase --abort	175
git rebase --continue	175
git rebase -i	170
git reflog	226
git remote	92
git remote prune	143
git remote update	109
git reset	43
git restore	40
git revert	42
git stash apply	85
git stash drop	85
git stash pop	85
git status	24
git submodule	232
git svn dcommit	211
git svn rebase	211
git tag	190
GUI	247

H

Hard-Reset	44
Hash	50
HEAD	56
detached	56
Historie	
Repository	34
Home-Verzeichnis	22
Hooks	248
Hotfix	191
Hotfix-Branch	191

I

id_rsa	122
id_rsa.pub	122

Ignore-Whitelist	88
Index	27
Installation	18
Windows	18
Interaktives Rebase	
Rebasing	166
Issue	118

J

Jenkins	145
---------------	-----

K

Konfiguration	23
---------------------	----

L

Lokale Versionsverwaltung	14
---------------------------------	----

M

Maintainer	
Workflow	142
master	56
Master-Branch	56
Mercurial	16, 203
Merge	
Fast-Forward	67
Recursive	68
Merge-Commit	69
Merge-Konflikt	69
Mergen	64
Strategien	73
Mixed-Reset	44

N

nano	33
Notepad++	33

O

Objekt	
blob	50
tree	50
Ordner	48
löschen	86
origin	95
origin/master	99

P

Parent-Objekt	51
Perforce	203

Projektverzeichnis	
säubern	85
Pull-Request	130

R

Rebase	
abbrechen	175
Rebasing	75
interaktives	166
recursive	74
Recursive-Merge	68
Reference Log	226
Reihenfolge	
Commits	173
Release-Branch	189
Remote-Repository	92
Benennung	94
konfigurieren	93
Repository	15
anlegen	21
anlegen (GitHub)	119
Binärdatei	205
Historie ansehen	34
klonen (GitHub)	128
konfigurieren (GitHub)	126
SVN	206
Reset	
Hard	44
Mixed	44
Soft	44
resolve	73

S

Schriftfarbe	38
Server	
zentraler	91
SHA-1	34
Shell	18
Soft-Reset	44
Squashen	175
automatisch	177
SSH-Agent	124
SSH-Key	122
Staging	27
Staging-Bereich	27
Änderung austragen	40
Stash	82

Submodul	233
Subversion	15, 203
svn add	205
svn branches	207
svn checkout	204
svn commit	204
svn copy	207
SVN-Repository	206
svn tags	207
switch	57

T

Tag	190
annotiert	190
leichtgewichtig	190
Namensgebung	191
Test	145
Themen-Branch	57
Topic-Branch	57
Tracking-Branch	103
tree-Objekt	50
trunk	207

U

Unmodified	32
Upstream-Branch	99

V

Version	
Definition	12
Unterschiede	18

Versionierte Datei	
verschieben	42
Versionskontrollprogramm	12
Versionsverwaltung	
lokale	14
verteilte	16
zentrale	15
Versionsverwaltungsprogramm	12
Verteilte Versionsverwaltung	16
vi	33

W

wget	25
Windows-Cmd	18
Workflow	165, 180
drei Repositories	109
Maintainer	142
mehrere Personen	183

Z

Zeile	
veränderte	38
Zentraler Server	91
Zentrale Versionsverwaltung	15
Zweig	55