O'REILLY®

2. Auflage

Git

kurz & gut

O'REILLYS TASCHENBIBLIOTHEK

Sven Riedel

2. AUFLAGE

Git

kurz & gut

Sven Riedel

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
E-Mail: kommentar@oreilly.de

1. Auflage 2009
2. Auflage 2014

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

Lektorat: Christine Haite, Volker Bombien, Köln
Fachliche Unterstützung: Florian Anderiasch, Michael Beck, München
Korrektorat: Tanja Feder
Satz: III-satz, Husby
Umschlaggestaltung: Michael Oreal, Köln
Produktion: Andrea Miß, Köln
Druck: fgb freiburger graphische betriebe, www.fgb.de

ISBN 978-3-95561-734-9

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

KAPITEL 1

Einleitung

Dieses Buch entstand, nachdem der Autor sich nach jahrelanger Verwendung von Subversion mit dem distributierten Versionierungssystem Git angefreundet hatte. Es soll keine erschöpfende Git-Referenz darstellen (was im Taschenbuchformat auch gar nicht möglich wäre). Vielmehr wird das Material im Referenzkapitel nach Aufgabenbereichen organisiert und dann beschrieben, wie bestimmte, mehr oder weniger alltägliche Aufgaben mit Git bewältigt werden können.

Was behandelt dieses Buch?

In diesem Buch wird Git tendenziell aus der Perspektive desjenigen betrachtet, der vornehmlich Subversion als Versionierungssystem verwendet oder verwendet hat. Insbesondere Kapitel 9, »Git-Client und Subversion-Server«, ist an Migrationswillige gerichtet, die sich zunächst einmal kontrolliert mit Git anfreunden wollen, bevor der vollständige Wechsel vollzogen wird, und für diejenigen Anwender, denen ein Subversion-Server vorgegeben ist, die aber dennoch die Vorzüge von Git nicht missen möchten.

Das bedeutet aber nicht, dass das Buch nur für diese Zielgruppe interessant wäre. Der übrige Teil besteht in einer Referenz zu den Alltagsbefehlen und deren meistgenutzten Parametern. Die einzelnen Git-Befehle bringen aber in der Regel einen weitaus größeren Umfang an Optionen und Parametern mit, als in diesem Buch auch nur ansatzweise umrissen werden könnte. Außerdem sind viele Low-Level-Befehle verfügbar, auf die hier gar nicht eingegangen werden kann. Falls Ihnen eine bestimmte Möglichkeit fehlt, emp-

fehle ich Ihnen, einen Blick in die Manpages der Git-Befehle zu werfen. In den meisten Fällen sollten Sie hier fündig werden.

Dennoch wird davon ausgegangen, dass Sie als Leser zumindest grundlegendes Wissen über die Unix-Umgebung und die Kommandozeile mitbringen und schon einmal mit einem Versionierungssystem gearbeitet haben.

Typographische Konventionen

Kursiv
: Kennzeichnet Pfadnamen und Dateinamen (z. B. Programmnamen), Internet-Adressen (z. B. Domainnamen und URLs) sowie hervorgehobene oder neu definierte Begriffe.

`Nichtproportionalschrift`
: Kennzeichnet Befehle und Optionen, die wörtlich eingegeben werden sollen.

`Nichtproportionalschrift kursiv`
: Kennzeichnet Werte, die vom Benutzer in entsprechend angepasst eingegeben werden.

`Nichtproportionalschrift fett`
: Wird verwendet, um Teile eines Programms oder einer Konfigurationsdatei hervorzuheben

Weitere Informationen und Hilfe

Um Git herum ist ein mächtiges Ökosystem mit sehr vielen Ressourcen im Netz vorhanden. Zentrale Anlaufstelle ist natürlich die Homepage von Git im WWW: *http://www.git-scm.com*. Hier finden Sie nicht nur die aktuellen Versionen zum Herunterladen, sondern auch ein Wiki sowie Links auf umfangreiche Dokumentationen für die verschiedensten Zielgruppen und auf verwandte Projekte.

Eine Liste von grafischen Frontends und Tools finden Sie unter *http://git.or.cz/gitwiki/InterfacesFrontendsAndTools*.

Freunde von Screencasts werden bei *http://git-scm.com/videos* glücklich.

KAPITEL 2

Konzepte

In diesem Kapitel werden die Grundkonzepte von Git in seiner Eigenschaft als verteiltes Versionsverwaltungssystem erörtert. Diese Konzepte sollten Sie kennen, wenn Sie die Arbeitsweise von Git im Detail verstehen wollen und den Grund kennen möchten, warum gerade eine bestimmte Vorgehensweise gewählt wird.

Der Arbeitsbaum, der Index und Commits

Ein Projekt spannt einen mehr oder weniger umfangreichen Verzeichnisbaum auf Ihrer Festplatte auf. Im zugehörigen Repository wird ein mehr oder minder großer Teil dieses Verzeichnisbaums verwaltet. Temporäre Dateien, Logdateien und andere transiente Daten werden üblicherweise aus der Verwaltung durch Git ausgeschlossen, obwohl sie Teil des Verzeichnisbaums des Projekts sind. Der von Git verwaltete Teil des Projekt-Verzeichnisbaums wird als `Arbeitsbaum` bezeichnet.

Wenn Git Änderungen in das Repository einbringen soll, erstellt Git zunächst ein virtuelles Abbild des aktuellen Zustands des Arbeitsbaumes. Dieser Prozess wird `staging` genannt, und das virtuelle Abbild ist in der Git-Nomenklatur der `Index`. Dies ist in der Abbildung dargestellt.

Beim Speichern der Änderungen in das Repository erzeugt Git aus dem aktuellen Index-Inhalt einen `Commit` und versieht diesen mit einer Revisionsnummer. Für die Nummerierung von Revisionen werden SHA1-Hashes verwendet, die aus dem Inhalt des Commits errechnet werden und aus 40 hexadezimalen Ziffern bestehen. Da

Git auch mit einem eindeutigen Präfix der Revisionsnummer arbeiten kann, wird üblicherweise nur mit den ersten 7 Ziffern gearbeitet.

Hier sieht man auch schon einen wesentlichen Unterschied von Git zu traditionellen Revisionsverwaltungssystemen wie CVS oder Subversion: Ein Commit bezieht sich auf den gesamten Arbeitsbaum und nicht auf einzelne Dateien. Git garantiert damit jederzeit einen konsistenten Zustand im Arbeitsbaum.

Sofern die Revisionsnummer bekannt ist, besteht unter Git zwar ebenfalls die Möglichkeit, einzelne Dateien auf den Zustand einer bestimmten Version zu setzen. Allerdings fließt dies als Änderung in den Zustand des aktuellen Arbeitsbaums ein. Dieser Arbeitsschritt ist unter Git eher unüblich und sollte nur mit Bedacht und in Einzelfällen durchgeführt werden.

Branches, Tags und Refs

Bei einem Branch handelt es sich um eine Kette von Commits, bei der jeder Commit auf seinen – oder seine – Eltern-Commit(s) verweist. Branches sind auch untereinander verknüpft; von jedem Commit kann ein neuer Branch abgezweigt werden, auch von solchen Commits, die inzwischen durch andere Commits verborgen werden. Ebenso können unterschiedliche Branches wieder zusammengeführt werden. Der Haupt-Branch heißt traditionell `master`, nimmt aber abgesehen von seinem Namen keinerlei Sonderstellung ein. Der jüngste Commit eines Branches wird `HEAD` genannt.

Das Zusammenführen von mehreren Branches wird als `merge` bezeichnet, und führt zur Erzeugung eines Merge-Commits im Repository. Ein Merge-Commit weist mehrere Eltern-Commits auf, die alle zueinander gleich gestellt sind. Beim Mergen agiert Git damit nach dem Prinzip »Merge die folgenden Commits, wobei der resultierende Commit Teil des aktuellen Branches werden soll« und nicht »Merge die folgenden Commits in den aktuellen Branch«. Der Unterschied ist subtil, wird aber deutlich, wenn ein Merge rückgän-

gig gemacht werden soll (siehe »Merges rückgängig machen« im Kapitel »Tipps und Tricks«.

Einzelne Commits können auch mit einem Namen versehen werden, damit auch künftig auf einfache Weise auf sie verwiesen werden kann. Dies wird als `tagging` bezeichnet und die Markierung entsprechend als `tag`.

Die Bezeichnung eines bestimmten Commits mittels seiner Revisionsnummer, des Tag-Namens, des Branch-Namens (was den `HEAD`-Commit impliziert) oder eine der anderen Möglichkeiten, die in Anhang aufgeführt sind, wird in der Git-Sprache als `Ref` bezeichnet.

Das Repository, Remote-Branches und Tracking-Branches

Das Git-Repository besteht im Wesentlichen aus einer Menge von Branches, die in einem Verhältnis zueinander stehen.

Als verteiltes Versionierungssystem bietet Git die Möglichkeit, Branches zwischen Ihrem Repository und anderen, entfernten Repositories zu lesen, zu schreiben und über das Netzwerk Branches miteinander zu synchronisieren. Ein Branch aus einem anderen Repository wird `remote branch` genannt.

Da Git als echtes dezentrales Versionierungssystem konzipiert wurde, ist die Topologie der Repositories untereinander nicht zwingend sternförmig aufgebaut; die Beziehungen der Repositories können einen beliebigen Graphen darstellen, wobei kein Repository auf technischer Ebene eine Sonderstellung im Hinblick auf andere Repositories einnimmt.

Aus diesem Grund muss Git den Zustand der entfernten Branches, mit denen Sie arbeiten wollen, lokal zwischenspeichern. Hierbei wird eine vollwertige, lokale Kopie des entfernten Branches angelegt, der die Grundlage für lokale Lese-, Schreib- und Synchronisierungsvorgänge darstellt. Dieser Cache des entfernten Branches wird als `tracking branch` bezeichnet.

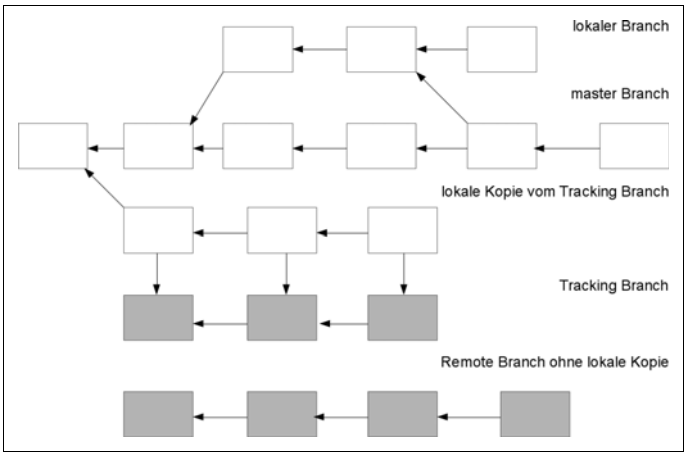

Abbildung 2-1: Repository mit Branches

KAPITEL 3

Installation

In diesem Kapitel wird beschrieben, wie Sie Git auf Ihrem Rechner installieren. Wie bei den meisten Open Source-Produkten können Sie entweder ein binäres Paket installieren, wie es vermutlich auch von der von Ihnen verwendeten Linux-Distribution mitgeliefert wird, oder sich den Quellcode besorgen und ihn manuell kompilieren. Binärpakete sind natürlich bequem und schnell, allerdings sind sie mit einem Eigenkompilat immer auf dem aktuellsten Stand.

Die zentrale Git-Website befindet sich aktuell unter *http://git-scm.com*. Hier finden Sie nicht nur den Quellcode und Binärpakete für die unterschiedlichen Betriebssysteme und Distributionen, sondern auch Dokumentationen, aktuelle Neuigkeiten und ein Wiki, das Ihnen bei Problemen vielerlei Art sehr helfen kann.

Binäre Installationspakete

Unter *http://git-scm.com/download* stehen binäre Installationspakete für OS X und Windows zur Verfügung.

Für Linux-Distributionen, FreeBSD, OpenBSD sowie Solaris finden sich hier die Installationsanleitungen, um das Git-Paket der jeweiligen Distribution zu installieren. Falls Ihr System hier nicht aufgeführt ist oder Sie mit der von Ihrer Distribution angebotenen Git-Version unzufrieden sind, können Sie auf die Kompilation und Installation aus dem Quelldateien ausweichen.

Linux

Der einfachste Weg besteht darin, das Git-Paket Ihrer verwendeten Linux Distribution zu installieren. Falls Sie unabhängig von Ihrer Distribution auf der Höhe der Zeit bleiben wollen, können Sie Git auch aus den Quell-Paketen heraus kompilieren und installieren.

OS X

Unter OS X stehen verschiedene Möglichkeiten zur Verfügung, Git zu installieren. Zum einen wird eine (meist etwas ältere) Git-Version mit Xcode ausgeliefert. Alternativ kann Git mittels Homebrew installiert und aktuell gehalten werden.

Die Installationsanleitung zu Homebrew findet man auf der entsprechenden Homepage: *http://brew.sh*.

Wenn Homebrew installiert ist, sollten zunächst die Paketdefinitionen mit

```
$ brew update
```

aktualisiert werden. Anschließend lässt sich Git ganz einfach mittels

```
$ brew install git
```

installieren. Mit

```
$ brew upgrade git
```

kann Git dann auf dem System aktuell gehalten werden.

TIPP

Wenn Sie sowohl Xcode als auch Git über Homebrew installiert haben, sollten Sie darauf achten, dass bei `PATH` der Installationspfad des Git aus Homebrew (*/usr/local/git/bin*) vor */usr/bin* aufgeführt ist. Anderenfalls wird Git nur aus dem Xcode-Paket genutzt.

Windows

Unter Windows können Sie Git in zwei Varianten installieren: als Cygwin-Paket oder als natives msys-Paket.

Gehen wir zunächst auf das msys-Paket ein, da hier die Installation recht gradlinig verläuft: Laden Sie sich das Installationspaket herunter, führen Sie es aus und freuen Sie sich über Git unter Windows. Das war's.

Um die Cygwin-Variante verwenden zu können, muss bereits die Cygwin-Umgebung installiert sein. Die können Sie unter *http://www.cygwin.com* beziehen. Unter Cygwin sollten Sie zusätzlich noch folgende Pakete installieren: `openssh` aus dem Abschnitt `Net` der Paketauswahl, aus dem `Libs`-Abschnitt `tcltk` sowie `subversion-perl` und zusätzlich einen Editor, den Sie für Commit-Kommentare und Ähnliches verwenden wollen.

Andere Betriebssysteme

Wenn Sie Git auf einem hier nicht aufgeführten Betriebssystem verwenden möchten, ohne es selbst kompilieren zu müssen, besteht noch eine weitere Möglichkeit – sofern eine Java-Laufzeitumgebung für Ihr Betriebssystem vorhanden ist. In der Eclipse-IDE ist eine reine Java-Implementierung eines Git-Clients integriert.

Selbst kompilieren

Das Kompilieren von Hand ist bei Git nicht sonderlich aufwendig, sofern Sie nur die Software an sich installieren möchten. Um die Dokumentation aus dem Quellcodepaket mit zu erstellen, sind einige zusätzliche Handgriffe erforderlich.

Quellcodepakete können als .zip- oder .tar.gz-Datei von *https://github.com/git/git/releases* heruntergeladen werden. Wenn Sie Git bereits einsetzen, können Sie natürlich auch das Repository für Git klonen und hieraus komplieren. Zur Entstehungszeit dieses Buches findet sich das autoritative Repository für Git auf GitHub unter *https://github.com/git/git*.

Voraussetzungen

Um Git erfolgreich zu kompilieren, sind folgende Bibliotheken und die entsprechenden Header-Dateien notwendig: zlib und SSH.

Perl 5.8 oder eine spätere Version wird ebenfalls empfohlen. Falls Sie nicht mit partiellen Commits arbeiten möchten oder mit SVN Repositories interagieren wollen, können Sie Perl beim Kompilieren umgehen, indem Sie beim *make*-Befehl den Parameter `NO_PERL=YesPlease` mit angeben. Im Regelfall sollten Sie Perl als Abhängigkeit allerdings belassen.

Optional, aber empfohlen ist OpenSSL für den SHA1-Hashing-Algorithmus. Falls Ihnen OpenSSL nicht zur Verfügung steht, wird Git auch mit einer alternativen Bibliothek für das Hashing ausgeliefert. Unter MacOS X versucht Git ab der Version 1.8.3 mit CommonCrypto anstatt OpenSSL zu linken. Wenn CommonCrypto nicht zur Verfügung steht, fällt Git auf OpenSSL zurück.

Falls Sie mit HTTP Projekte verwalten möchten, benötigen Sie noch `libcurl` und `expat`. Wenn Ihnen das SSH- sowie das Git-native Protokoll genügen, sind diese Bibliotheken nicht erforderlich.

Für den Fall, dass Sie die grafischen Tools verwenden möchten, benötigen Sie noch das `wish`-Paket für Tcl/TK.

Eine `gettext`-Bibliothek ist für lokalisierte Git-Ausgaben erforderlich. Im Regelfall wird mit GNU's `libintl` gearbeitet, allerdings funktioniert auch die standardmäßige `gettext`-Bibliothek von Solaris. Wenn Sie keine übersetzten Texte verwenden möchten, können Sie bei der Kompilation dem *make*-Befehl den Parameter `NO_GETTEXT=YesPlease` übergeben. In diesem Fall wird Git lediglich englischsprachige Ausgaben bereitstellen.

Sofern Sie die Perforce-Konnektivität von Git verwenden möchten, sollten Sie Python 2.4 oder eine spätere Version bereitstellen. Python 3.x wird hier jedoch aktuell nicht unterstützt.

Wenn Sie die Hilfstexte und Dokumentation ebenfalls generieren und installieren möchten, muss *AsciiDoc* installiert sein. Beim Autor hat das zu einer Kaffeepause und der Installation von knapp

300 MByte an Abhängigkeiten geführt. Alternativ können Sie vorformatierte Hilfstexte separat herunterladen und installieren. Dies wird im Abschnitt zur Generierung der Hilfstexte in diesem Kapitel beschrieben (siehe »Installieren der Hilfstexte und Manpages« auf Seite 20).

AsciiDoc baut auf DocBook auf. Es gibt Berichte darüber, dass die DocBook XSL-Definitionen der Version 1.72 und 1.73 fehlerhaft seien. Für 1.73 müssen Sie gegebenenfalls den mitgelieferten Patch *contrib/patches/docbook-xsl-manpages-charmap.patch* installieren.

Das Kompilieren

Den Quellcode können Sie als traditionelles Archiv unter *https://github.com/git/git/releases* herunterladen und mit

```
$ tar xfzv git-X.Y.Z.tar.gz
```

bzw.

```
$ unzip git-X.Y.Z.zip
```

entpacken, wobei es sich bei `X.Y.Z` um die Version des heruntergeladenen Git-Paketes handelt. Falls bei Ihrem Betriebssystem kein Gnu-Tar mitgeliefert wird, werden Sie wahrscheinlich das Paket zunächst explizit entpacken müssen:

```
$ gzip -c git-X.Y.Z.tar.gz | tar -xvf -
```

Alternativ können Sie die aktuellen Git-Quellen auch mit Git herunterladen, sofern Sie es bereits installiert haben:

```
$ git clone https://github.com/git/git.git
```

Wechseln Sie in das Verzeichnis mit dem gerade entpackten Git-Quellcode und führen Sie die Befehle

```
$ make
$ make install
```

aus, um Git (zunächst ohne Dokumentation) mit den Standard-Konfigurationsoptionen zu installieren.

Da Git von Hause aus kein *configure*-Skript mehr zur Verfügung stellt, müssen Sie Zielverzeichnisse im Makefile direkt angeben.