

Git

Dezentrale Versionsverwaltung im Team - Grundlagen und Workflows

von

René Preißel, Bjørn Stachmann

2., aktualisierte und erweiterte Auflage

Git – Preißel / Stachmann

schnell und portofrei erhältlich bei beck-shop.de DIE FACHBUCHHANDLUNG

Thematische Gliederung:

Software Engineering

dpunkt.verlag 2013

Verlag C.H. Beck im Internet:

www.beck.de

ISBN 978 3 86490 130 0

Inhaltsverzeichnis

Erste Schritte

1	Grundlegende Konzepte	1
1.1	Dezentrale Versionsverwaltung – alles anders?	1
1.2	Das Repository – die Grundlage dezentralen Arbeitens	3
1.3	Branching und Merging – ganz einfach!	5
1.4	Zusammenfassung	7
2	Erste Schritte	9
2.1	Git einrichten	9
2.2	Das erste Projekt mit Git	9
2.3	Zusammenarbeit mit Git	13
2.4	Zusammenfassung	18

Arbeiten mit Git

3	Was sind Commits?	19
3.1	Zugriffsberechtigungen und Zeitstempel	19
3.2	Die Befehle add und commit	20
3.3	Exkurs: Mehr über Commit-Hashes	20
3.4	Eine Historie von Commits	21
3.5	Eine etwas andere Sichtweise auf Commits	22
3.6	Viele unterschiedliche Historien desselben Projekts	23
3.7	Zusammenfassung	26
4	Commits zusammenstellen	27
4.1	Der Status-Befehl	27
4.2	Der Stage-Bereich speichert Momentaufnahmen	30
4.3	Was tun mit Änderungen, die nicht übernommen werden sollen?	32
4.4	Mit .gitignore Dateien unversioniert lassen	34
4.5	Stashing: Änderungen zwischenspeichern	35
4.6	Zusammenfassung	36

5	Das Repository	37
5.1	Ein einfaches und effizientes Speichersystem	37
5.2	Verzeichnisse speichern: Blob & Tree	38
5.3	Gleiche Daten werden nur einmal gespeichert	38
5.4	Kompression ähnlicher Inhalte	39
5.5	Ist es schlimm, wenn verschiedene Daten zufällig denselben Hashwert bekommen?	39
5.6	Commits	40
5.7	Wiederverwendung von Objekten in der Commit-Historie	40
5.8	Umbenennen, verschieben und kopieren	41
5.9	Zusammenfassung	44
6	Branches verzweigen	45
6.1	Parallele Entwicklung	45
6.2	Bugfixes in älteren Versionen	46
6.3	Branches	46
6.4	<i>Swimlanes</i>	47
6.5	Aktiver Branch	47
6.6	Branch-Zeiger umsetzen	49
6.7	Branch löschen	50
6.8	Und was ist, wenn man die Commit-Objekte wirklich loswerden will?	51
6.9	Zusammenfassung	52
7	Branches zusammenführen	53
7.1	Was passiert bei einem Merge?	54
7.2	Konflikte	55
7.3	Bearbeitungskonflikte	56
7.4	Konfliktmarkierungen	56
7.5	Bearbeitungskonflikte lösen	57
7.6	Und was ist mit den inhaltlichen Konflikten?	59
7.7	Fast-Forward-Merges	59
7.8	First-Parent-History	61
7.9	Knifflige Merge-Konflikte	62
7.10	Egal, es wird schon irgendwie gehen	63
7.11	Zusammenfassung	64
8	Mit Rebasing die Historie glätten	65
8.1	Das Prinzip: Kopieren von Commits	65
8.2	»Diamantenketten« vermeiden	66
8.3	Und wenn es zu Konflikten kommt?	67
8.4	Branches umpflanzen	68

8.5	Was passiert mit den ursprünglichen Commits nach dem Re-basing?	70
8.6	Warum ist es problematisch, Original und Kopie eines Com-mits im gleichen Repository zu haben?	70
8.7	Cherry-Picking	71
8.8	Zusammenfassung	71
9	Austausch zwischen Repositories	73
9.1	Repositories klonen	73
9.2	Wie sagt man Git, wo das andere Repository liegt?	74
9.3	Anderen Repositorys einen Namen geben	75
9.4	Abholen von Daten	75
9.5	Remote-Tracking-Branched: Wissen, was in anderen Reposi-torys »los« ist	77
9.6	Lokal mit Branches aus anderen Repositorys arbeiten	78
9.7	<i>Pull = Fetch + Merge</i>	78
9.8	Für Diamantenhasser: --rebase	79
9.9	<i>Push</i> – das Gegenstück zu <i>Pull</i>	79
9.10	Jeder so, wie er mag	81
9.11	Zusammenfassung	81
10	Versionen markieren	83
10.1	Arbeiten mit Tags erstellen	83
10.2	Welche Tags gibt es?	84
10.3	Die <i>Hashes</i> zu den Tags ausgeben	84
10.4	Die Log-Ausgaben um Tags anreichern	85
10.5	In welcher Version ist es »drin«?	85
10.6	Wie verschiebt man ein Tag?	85
10.7	Und wenn ich ein »Floating Tag« brauche?	86
10.8	Zusammenfassung	86
11	Abhängigkeiten zwischen Repositories	87
11.1	Abhängigkeiten mit Submodulen	87
11.2	Abhängigkeiten mit Subtrees	93
11.3	Zusammenfassung	98
12	Tipps und Tricks	101
12.1	Keine Panik – Es gibt ein <i>Reflog</i> !	101
12.2	Lokale Änderungen temporär ignorieren	102
12.3	Änderungen an Textdateien untersuchen	103
12.4	<i>alias</i> – Abkürzungen für Git-Befehle	104
12.5	Branches als temporäre Zeiger auf Commits nutzen	104
12.6	Commits auf einen anderen Branch verschieben	105

Workflows

13 **Workflow-Einführung** 107

13.1 Warum Workflows? 107

13.2 Welche Workflows sind wann sinnvoll? 108

13.3 Aufbau der Workflows 109

Workflows: Entwickeln mit Git

14 **Ein Projekt aufsetzen** 111

14.1 Ablauf und Umsetzung 114

14.2 Warum nicht anders? 125

15 **Gemeinsam auf einem Branch entwickeln** 127

15.1 Ablauf und Umsetzung 130

15.2 Warum nicht anders? 132

16 **Mit Feature-Banches entwickeln** 135

16.1 Ablauf und Umsetzung 137

16.2 Warum nicht anders? 146

17 **Mit Bisection Fehler suchen** 153

17.1 Ablauf und Umsetzung 155

17.2 Warum nicht anders? 162

Workflows: Entwicklungsprozess

18 **Mit einem Build-Server arbeiten** 165

18.1 Ablauf und Umsetzung 168

18.2 Warum nicht anders? 176

19 **Ein Release durchführen** 179

19.1 Ablauf und Umsetzung 182

19.2 Warum nicht anders? 189

Workflows: Repositorys pflegen

20 **Große Projekte aufteilen** 191

20.1 Ablauf und Umsetzung 194

20.2 Warum nicht anders? 198

21	Kleine Projekte zusammenführen	199
21.1	Ablauf und Umsetzung	202
21.2	Warum nicht anders?	204
22	Lange Historien auslagern	205
22.1	Ablauf und Umsetzung	208
22.2	Warum nicht anders?	213

Workflows: Umstieg auf Git

23	Andere Versionsverwaltungen parallel nutzen	215
23.1	Ablauf und Umsetzung	218
23.2	Warum nicht anders?	225
24	Ein Projekt nach Git migrieren	227
24.1	Ablauf und Umsetzung	230
24.2	Warum nicht anders?	240

Mehr über Git

25	Was gibt es sonst noch?	243
25.1	Interaktives Rebasing – Historie verschönern	243
25.2	Umgang mit Patches	244
25.3	Patches per Mail versenden	244
25.4	Bundles – Pull im Offline-Modus	245
25.5	Archive erstellen	246
25.6	Grafische Werkzeuge für Git	246
25.7	Repository im Webbrowser anschauen	247
25.8	Zusammenarbeit mit Subversion	247
25.9	Aliase für Befehle	248
25.10	Notizen an Commits	249
25.11	Hooks – Git erweitern	249
25.12	Github – Hosting von Repositorys	249
26	Das sechszwanzigste Kapitel	251

27 Grenzen von Git 253
27.1 Hohe Komplexität 253
27.2 Komplizierter Umgang mit Submodulen 254
27.3 Ressourcenverbrauch bei großen binären Dateien 255
27.4 Repositorys können nur vollständig verwendet werden 256
27.5 Autorisierung nur auf dem ganzen Repository 257
27.6 Mäßige grafische Werkzeuge für Historienauswertung 258

Index und Verzeichnisse

»Schritt für Schritt«-Anleitungen 260

Workflow-Verzeichnis 263

Index 269