

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	22
3.7	Zusammenfassung	25
4	Commits zusammenstellen	27
4.1	Der Status-Befehl	27
4.2	Stage-Bereich speichert Momentaufnahmen	30
4.3	Was tun mit Änderungen, die nicht übernommen werden sollen?	32
4.4	Mit .gitignore Dateien unversioniert lassen	33
4.5	Stashing: Änderungen zwischenspeichern	33
4.6	Zusammenfassung	35

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	39
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	Branchzeiger 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?	58
7.7	Fast-Forward-Merges	59
7.8	Knifflige Merge-Conflikte	60
7.9	Ach egal, wird schon irgendwie gehen	62
7.10	Zusammenfassung	62
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 Rebasing?	70

8.6	Warum ist es problematisch, Original und Kopie eines Commits im gleichen Repository zu haben?	70
8.7	Cherry-Picking	71
8.8	Zusammenfassung	71
9	Austausch zwischen Repositorys	73
9.1	Repositorys 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-Banches: Wissen, was in anderen Repositorys »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, 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	Log-Ausgabe, um <i>Tags</i> anreichern	85
10.5	In welcher Version ist es »drin«?	85
10.6	Wie verschiebt man ein <i>Tag</i> ?	85
10.7	Und wenn ich ein »Floating Tag« brauche?	86
10.8	Zusammenfassung	86
11	Submodule	87
11.1	Submodule	87
11.2	Zusammenfassung	93

Workflows

12	Workflow-Einführung	95
12.1	Warum Workflows?	95
12.2	Welche Workflows sind wann sinnvoll?	96
12.3	Aufbau der Workflows	97

Workflows: Entwickeln mit Git

13	Ein Projekt aufsetzen	99
13.1	Ablauf und Umsetzung	102
13.2	Warum nicht anders?	113
14	Gemeinsam auf einem Branch entwickeln	115
14.1	Ablauf und Umsetzung	118
14.2	Warum nicht anders?	120
15	Mit Feature-Banches entwickeln	123
15.1	Ablauf und Umsetzung	125
15.2	Warum nicht anders?	134
16	Mit Bisection Fehler suchen	141
16.1	Ablauf und Umsetzung	143
16.2	Warum nicht anders?	150

Workflows: Entwicklungsprozess

17	Mit einem Build-Server arbeiten	153
17.1	Ablauf und Umsetzung	156
17.2	Warum nicht anders?	165
18	Ein Release durchführen	167
18.1	Ablauf und Umsetzung	170
18.2	Warum nicht anders?	177

Workflows: Repositorys pflegen

19	Große Projekte aufteilen	179
19.1	Ablauf und Umsetzung	182
19.2	Warum nicht anders?	185
20	Kleine Projekte zusammenführen	187
20.1	Ablauf und Umsetzung	190
20.2	Warum nicht anders?	192
21	Lange Historien auslagern	193
21.1	Ablauf und Umsetzung	196
21.2	Warum nicht anders?	201

Workflows: Umstieg auf Git

22	Andere Versionsverwaltungen parallel nutzen	203
22.1	Ablauf und Umsetzung	206
22.2	Warum nicht anders?	213
23	Ein Projekt nach Git migrieren	215
23.1	Ablauf und Umsetzung	218
23.2	Warum nicht anders?	228

Mehr über Git

24	Was gibt es sonst noch?	231
24.1	Interaktives Rebasing – Historie verschönern	231
24.2	Umgang mit Patches	232
24.3	Patches per Mail versenden	232
24.4	Bundles – Pull im Offline-Modus	233
24.5	Archive erstellen	234
24.6	Grafische Werkzeuge für Git	234
24.7	Repository im Webbrowser anschauen	235
24.8	Zusammenarbeit mit Subversion	235
24.9	Aliase für Befehle	236
24.10	Notizen an Commits	237
24.11	Hooks – Git erweitern	237
24.12	Github – Hosting von Repositorys	237
25	Grenzen von Git	239
25.1	Hohe Komplexität	239
25.2	Komplizierter Umgang mit Submodulen	240
25.3	Ressourcenverbrauch bei großen binären Dateien	241
25.4	Repositorys können nur vollständig verwendet werden	242
25.5	Autorisierung nur auf dem ganzen Repository	243
25.6	Mäßige grafische Werkzeuge für Historienauswertung	244

Index & Verzeichnisse

»Schritt für Schritt«-Anleitungen	245
Workflow-Verzeichnis	247
Index	253