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 mit der Kommandozeile 9
2.1 Git einrichten 9
2.2 Ein paar Hinweise für Windows-User 9
2.3 Git einrichten 11
2.4 Das erste Projekt mit Git 12
2.5 Zusammenarbeit mit Git 15
2.6 Zusammenfassung 21

3 Erste Schritte mit SourceTree 23
3.1 SourceTree konfigurieren 23
3.2 Das erste Projekt mit Git 23
3.3 Zusammenarbeit mit Git 26
3.4 Zusammenfassung 30

Arbeiten mit Git

4 Was sind Commits? 31
4.1 Zugriffsberechtigungen und Zeitstempel 32
4.2 Die Befehle `add` und `commit` 32
4.3 Exkurs: Mehr über Commit-Hashes 33
4.4 Eine Historie von Commits 34
4.5 Eine etwas andere Sichtweise auf Commits 34
4.6 Viele unterschiedliche Historien desselben Projekts 36
4.7 Zusammenfassung 38

5	**Commits zusammenstellen**	**39**
5.1	Der `status`-Befehl	39
5.2	Der Stage-Bereich speichert Momentaufnahmen	43
5.3	Was tun mit Änderungen, die nicht übernommen werden sollen?	45
5.4	Mit `.gitignore` Dateien unversioniert lassen	46
5.5	Stashing: Änderungen zwischenspeichern	47
5.6	Zusammenfassung	48
6	**Das Repository**	**49**
6.1	Ein einfaches und effizientes Speichersystem	49
6.2	Verzeichnisse speichern: Blob und Tree	50
6.3	Gleiche Daten werden nur einmal gespeichert	51
6.4	Kompression ähnlicher Inhalte	51
6.5	Ist es schlimm, wenn verschiedene Daten zufällig denselben Hashwert bekommen?	52
6.6	Commits	52
6.7	Wiederverwendung von Objekten in der Commit-Historie	53
6.8	Umbenennen, verschieben und kopieren	54
6.9	Zusammenfassung	56
7	**Branches verzweigen**	**59**
7.1	Parallele Entwicklung	59
7.2	Bugfixes in älteren Versionen	60
7.3	Branches	60
7.4	Aktiver Branch	61
7.5	Branch-Zeiger umsetzen	64
7.6	Branch löschen	64
7.7	Und was ist, wenn man die Commit-Objekte wirklich loswerden will?	66
7.8	Zusammenfassung	66
8	**Branches zusammenführen**	**67**
8.1	Was passiert bei einem Merge?	68
8.2	Konflikte	69
8.3	Fast-Forward-Merges	74
8.4	First-Parent-History	75
8.5	Knifflige Merge-Konflikte	76
8.6	Zusammenfassung	78
9	**Mit Rebasing die Historie glätten**	**81**
9.1	Das Prinzip: Kopieren von Commits	81
9.2	Und wenn es zu Konflikten kommt?	83

9.3 Was passiert mit den ursprünglichen Commits nach dem Rebasing? ... 84
9.4 Empfehlungen zum Rebasing ... 85
9.5 Cherry-Picking ... 88
9.6 Zusammenfassung ... 88

10 Repositorys erstellen, klonen und verwalten ... 89
10.1 Ein Repository erstellen ... 89
10.2 Das Repository-Layout ... 89
10.3 Bare-Repositorys ... 90
10.4 Vorhandene Dateien übernehmen ... 90
10.5 Ein Repository klonen ... 91
10.6 Wie sagt man Git, wo das Remote-Repository liegt? ... 91
10.7 Kurznamen für Repositorys: Remotes ... 92
10.8 Zusammenfassung ... 93

11 Austausch zwischen Repositorys ... 95
11.1 Fetch, Pull und Push ... 95
11.2 Remote-Tracking-Branches ... 96
11.3 Einen Remote-Branch bearbeiten ... 97
11.4 Ein paar Begriffe, die man kennen sollte ... 98
11.5 Fetch: Branches aus einem anderen Repository holen ... 99
11.6 Fetch: Aufrufvarianten ... 99
11.7 Erweiterte Möglichkeiten ... 104
11.8 Zusammenfassung ... 104

12 Versionen markieren ... 105
12.1 Arbeiten mit Tags erstellen ... 105
12.2 Welche Tags gibt es? ... 106
12.3 Die Hashes zu den Tags ausgeben ... 106
12.4 Die Log-Ausgaben um Tags anreichern ... 107
12.5 In welcher Version ist es »drin«? ... 107
12.6 Wie verschiebt man ein Tag? ... 107
12.7 Und wenn ich ein »Floating Tag« brauche? ... 108
12.8 Zusammenfassung ... 108

13 Tipps und Tricks ... 109
13.1 Keine Panik – es gibt ein Reflog! ... 109
13.2 Lokale Änderungen temporär ignorieren ... 110
13.3 Änderungen an Textdateien untersuchen ... 111
13.4 `alias` – Abkürzungen für Git-Befehle ... 112
13.5 Branches als temporäre Zeiger auf Commits nutzen ... 113
13.6 Commits auf einen anderen Branch verschieben ... 114
13.7 Mehr Kontrolle bei Fetch, Push und Pull ... 115

Workflows

14 Workflow-Einführung 117
14.1 Warum Workflows? 117
14.2 Welche Workflows sind wann sinnvoll? 118
14.3 Aufbau der Workflows 119

Workflows: Entwickeln mit Git

15 Ein Projekt aufsetzen 123

16 Gemeinsam auf einem Branch entwickeln 135

17 Mit Feature-Branches entwickeln 143

18 Mit Forks entwickeln 163

Workflows: Release-Prozess

19 Kontinuierlich Releases durchführen 175

20 Periodisch Releases durchführen 185

21 Mit mehreren aktiven Releases arbeiten 199

Workflows: Repositorys pflegen

22 Ein Projekt mit großen binären Dateien versionieren 213

23 Große Projekte aufteilen 221

24 Kleine Projekte zusammenführen 229

25 Lange Historien auslagern 235

26 `http://kapitel26.github.io` 245

27 Ein Projekt nach Git migrieren 247

Mehr über Git

28 Integration mit Jenkins ... 263
28.1 Vorbereitungen ... 263
28.2 Ein einfaches Git-Projekt einrichten ... 264
28.3 Hook als Build-Auslöser ... 265
28.4 Ein Tag für jeden erfolgreichen Build ... 267
28.5 Pull-Requests bauen ... 269
28.6 Automatischer Merge von Branches ... 272

29 Abhängigkeiten zwischen Repositorys ... 275
29.1 Abhängigkeiten mit Submodulen ... 275
29.2 Abhängigkeiten mit Subtrees ... 281
29.3 Zusammenfassung ... 286

30 Was gibt es sonst noch? ... 289
30.1 Worktrees – mehrere Workspaces mit einem Repository ... 289
30.2 Interaktives Rebasing – Historie verschönern ... 290
30.3 Umgang mit Patches ... 291
30.4 Archive erstellen ... 291
30.5 Grafische Werkzeuge für Git ... 292
30.6 Repository im Webbrowser anschauen ... 293
30.7 Zusammenarbeit mit Subversion ... 294
30.8 Hooks – Git erweitern ... 294
30.9 Mit Bisection Fehler suchen ... 294

31 Die Grenzen von Git ... 297
31.1 Hohe Komplexität ... 297
31.2 Komplizierter Umgang mit Submodulen ... 299
31.3 Ressourcenverbrauch bei großen binären Dateien ... 300
31.4 Repositorys können nur vollständig verwendet werden ... 300
31.5 Autorisierung nur auf dem ganzen Repository ... 301
31.6 Mäßige grafische Werkzeuge für die Historienauswertung ... 302

Anhang

Schritt-für-Schritt-Anleitungen ... 305

Workflow-Verzeichnis ... 307

Index ... 313