Title: Arbeiten mit Git Date: 2014-01-28 Author: Georg Wirtenberger

1. Einleitung

Dieses Tutorial gibt eine kurzen Einblick in Git. Zuerst werden Ihnen die Grundlagen, die zum Verstehen und Arbeiten mit Git nötig sind, vorgestellt. Danach wird anhand eines Beispiels erklärt, wie man ein Projekt mit Hilfe von Git aufsetzt. Zum Schluss werden noch einige Workflow, die man mit Git realisieren kann, vorgestellt.

---

2. Grundlagen

In diesem Kapitel erfahren Sie was Git ist und wie es arbeitet. Es wird hier extra auf Details zur Arbeitsweise verzichtet, denn es würde den Rahmen dieses Tutorials sprengen. Es wird nur das was zum Verstehen und Arbeiten mit Git nötig ist erwähnt.

2.1 Versionsverwaltung

Wenn man an einem Projekt arbeitet, dann möchte man vermeiden, dass die Daten verloren gehen und manchmal möchte man zu einem älteren Stand zurück kehren. Damit dies einfach, schnell und möglichst ohne Probleme möglich ist, wurden einige Konzepte zur Versionsverwaltung entwickelt. Wahrscheinlich benutzen Sie auch eine Art der Versionsverwaltung, indem Sie Kopien von Ihren Projekten erstellen und diese entweder lokal oder auf einem externen Datenträger ab speichern. Eine Software zur Versionsverwaltung übernimmt diese Arbeitsschritte für Sie. Es gibt einige Konzepte für die Versionsverwaltung, die Ihre Vor-und Nachteile haben. In diesem Tutorial erfahren Sie über eines davon, nämlich Git. Es wird hier extra auf die Vor-und Nachteile von Git gegenüber anderen Konzepten verzichtet. Denn, wenn Sie dieses Tutorial lesen, dann haben Sie sich schon für Git entschieden.

2.2 Snapshots

Um verschiedene Versionen zu speichern, benutzt Git Snapshots. Snapshots sind Momentaufnahmen des Zustandes Ihres Projektes. Wenn Git einen Snapshot erstellt hat, dann können Sie jeder Zeit zu diesem Snapshot wechseln. In der Abbildung 1 sind die einzelnen Snapshots über die Zeit zu sehen.

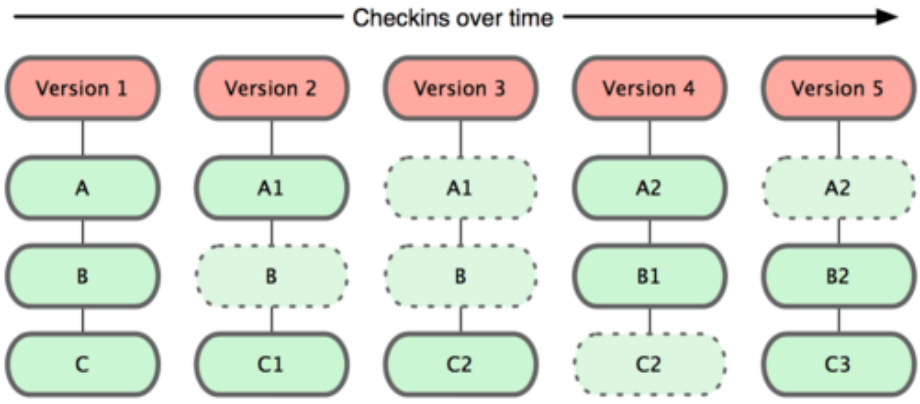

Abbildung 1: Snapshots über die Zeit

2.3 Drei Zustände

Git hat drei Zustände, in denen Ihre Daten sich befinden können: eingecheckt (commited), vorbereitet (staged) und verändert (modified).

- Eingecheckt bedeutet, dass die Daten in der lokalen Datenbank gespeichert sind.
- Vorbereitet heißt, dass die Daten bearbeitet und zum Einchecken vorbereitet wurden (in staging area verschoben wurden). Jede Datei, die Sie bearbeitet haben und Einchecken möchten, muss in die staging are verschoben werden, sonst wird sie nicht eingecheckt.
- Bei veränderten Daten handelt es sich um Daten, die verändert wurden, aber noch nicht in die staging area verschoben wurden.

Abbildung 2 zeigt diese Zustände.

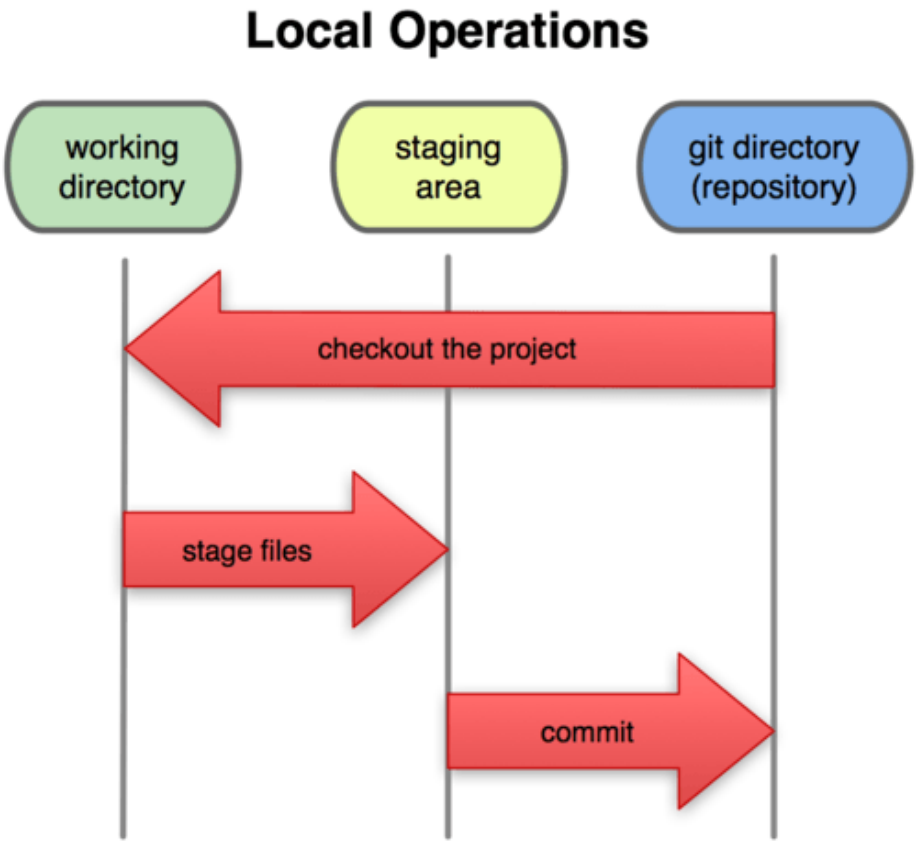

Abbildung 2: Drei Zustände

Git directory ist die Kopie der Repository, die sich lokal auf Ihrer Festplatte befindet. Die staging area ist eine Datei, die sich im Git Verzeichnis befindet. In

dieser Datei befinden sich die Information darüber, was beim nächsten Einchecken übertragen wird (Es ist der Snapshot der beim Einchecken gespeichert wird). Working directory ist eine Kopie einer aus der Datenbank geholten Version des Projektes (checkout).

Der standard Workflow mit Git sieht also folgendermaßen aus:

1. Sie bearbeiten die Daten
2. Sie fügen die bearbeiteten Daten der staging area zu
3. Sie checken die Daten ein

2.4 Dateistatus

Jede Datei, die sich in Ihrem Arbeitsverzeichnis befindet, hat zwei Zustände: entweder verfolgt(tracked) oder oder nicht verfolgt (untracked).

- Verfolgt bedeutet, dass die Datei beim letzte Einchecken sich im Snapshot befand.
- Nicht verfolg heißt, dass diese Datei neu zum Projekt gekommen ist. Sie muss also dem Projekt noch hinzugefügt werden. Sie wird dem Projekt hinzugefügt, in dem Sie sie zur staging area hinzufügen.

In der Abbildung 3 ist der Dateistatus Lebens Zyklus zusehen.

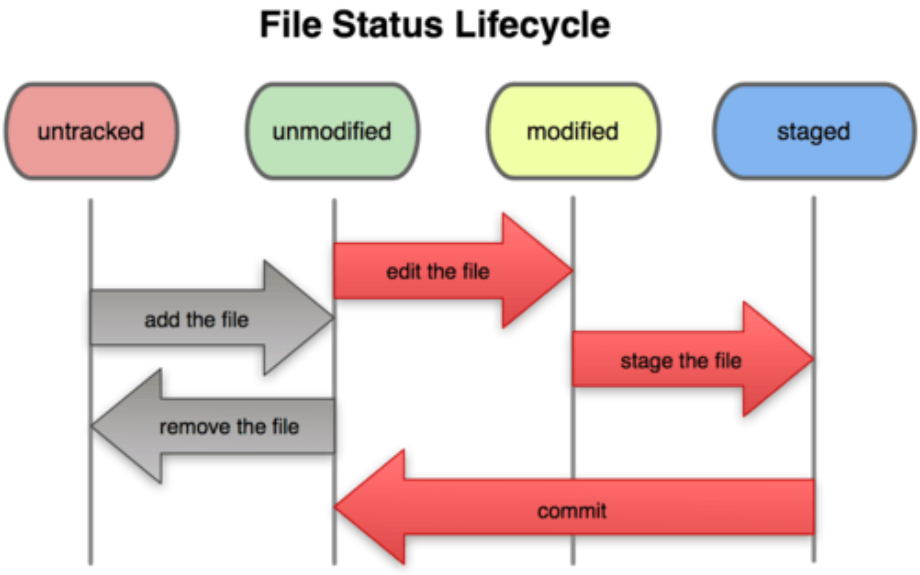

Abbildung 3: Dateistatus Lebenszyklus

Wenn die Datei nicht verändert wurde, betrachtet Git diese als nicht bearbeitet. Sobald Sie Änderungen an der Datei vornehmen, gilt sie für Git als bearbeitet

und Sie müssen diese Datei zur stage area hinzufügen, damit sie eingecheckt werden kann. Danach wird das Ganze wiederholt.

2.5 Branches

Beim Programmieren kann es vorkommen, dass Sie etwas Neues ausprobieren möchte oder Sie müssen einen Fehler korregieren. Damit Sie es machen können ohne das laufende Programm zu beschädigen, gibt es bei Git die Möglichkeit einen Zweig (Branch) zu erstellen, an dem Zweig zu arbeiten und dann den Code wieder zusammen zu führen.

Wenn Sie mit Git ein Repository anlegen, dann erzeugt Git einen Standardzweig, der Master heisßt. Sie arbeiten dann am Master und erstellen dort Ihre Commits. Der Masterzweig ist Ihr aktuelle laufendes Programm, das Sie vielleicht schon ausgeliefert haben. Wenn Sie z.B. einen Teil Ihres Programms optimeiren wollen, ohne dabei das laufende Programm kaputt zu machen, dann erstellen Sie einfach einen anderen Zweig und arbeiten an diesem. Abbildung 4 zeigt einen Beispiel der Verzweigung.

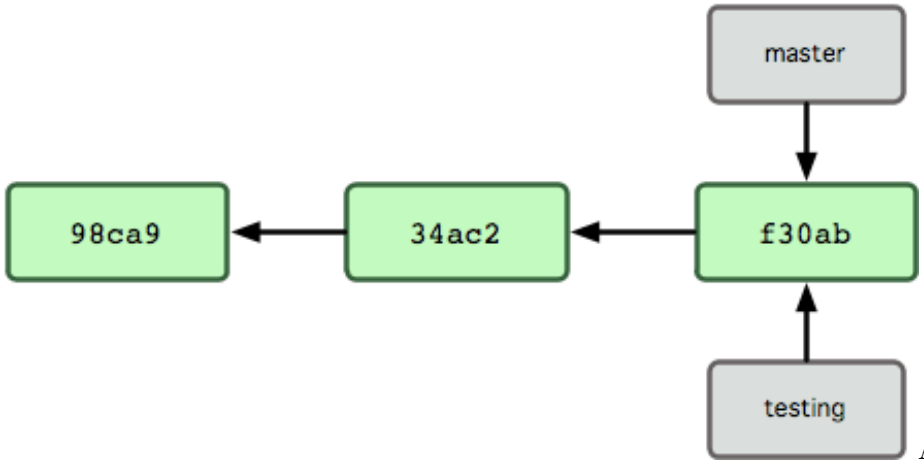

Abbildung 4: Beispiel einer Verzweigung

Wenn Sie mit Ihrem Code zufrieden sind, dann können Sie diesen Zweig und Masterzweig zusammenführen. Den Zweig, an dem Sie gerade gearbeitet haben können Sie stehen lassen oder Sie löschen ihn.

---

3. Arbeiten mit Git

In diesem Abschnitt wird Ihnen Schritt für Schritt erklärt, wie man mit Git arbeitet. Es wird dabei ein Beispielprojekt erstellt. Anhand diese Projektes wer-

den die wichtigsten Befehle, die Sie zum Arbeiten mit Git brauchen, vorgestellt. Es wird erklärt, wie Sie ein Repository erzeugen, eine Kopie (Clone) davon erstellen, Änderungen an dieser Kopie machen und alles wieder zusammenführen. Außerdem wird das Arbeiten mit Branches vorgestellt.

Es wird das Arbeiten mit Git mit Hilfe der Kommandozeile vorgestellt. Wenn sie ein anderes Tool zum arbeiten mit Git benutzen z.B.: GitHub, dann sollte es Ihnen nicht schwer fallen, dieses Tool zu erlernen, denn die Begriffe sollten bei allen Tools gleich sein.

Repository erstellen

Wenn Sie ein Repository mit Git erstellen möchten, dann wächseln Sie in das Projektcerzeichnis und für *git init* aus.

```
$ git init
```

Dieser Befehl erzeugt ein Unterverzeichnis *.git*, das alle für Git relevanten Daten enthält. Dieser Verzeichnis ist versteckt. Zu diesem Zeitpunkt befinden sich die Daten im Verzeichnis im Zustand "nicht verfolgt" (untracked). Um die Daten in den Zustand "verfolgt" (tracked) zu überführen, muss man sie dem Index hinzufügen. Wie das geht wird später in diesem Kapitel erklärt.

Repository clonen

Wenn Sie sich ein Kopie der bereits bestehenden Repository erstellen möchten, dann müssen Sie folgenden Befehl *git clone [url]* ausführen.

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

3.1 Arbeiten am Clone

Sobald Sie sich eine Kopie der Repository erstellt haben, dann haben Sie zwei Möglichkeiten an dieser Kopie zu arbeiten. Sie können am Masterzweig arbeiten oder sich einen extra Zweig erstellen. Dabei ist die Arbeitsweise am Master und Branches gleich.

3.1.1 Arbeiten am Master

Wenn Sie am Masterzweig arbiten, dann sieht der Arbeitsablauf ungefähr so aus:

- Sie bearbeiten die Daten
- Sie fügen die daten zum Index hinzu (track)

```
$ git add *.js
```

- Sie entfernen die Daten aus der Repository

```
$ rm test.js
```

- Sie cheken die Daten ein (commiten)

```
$ git commit -m 'alle Javascriptdateien wurden eingechekt'
```

- Sie wechseln zur vorherigen Version

```
$ git checkout [Hashwert des Commits]
```

3.1.2 Arbeiten an Branches (Zweigen)

Wenn Sie sich entschieden haben an einem Branch zu arbeiten, weil Sie z.B. etwas Neues ausprobieren möchten, dann sieht Ihre Vorgehnsweise so aus.

Branch erstellen

Sie erstellen einen Branch und wechseln zu Demselbigen mit folgenden Befehlen:

```
$ git branch myBranch
$ git checkout myBranch
```

Nun können Sie wie gewohnt mit Daten arbeiten.

Arbeiten mit Daten

- Sie bearbeiten die Daten
- Sie fügen die daten zum Index hinzu (track)

```
$ git add *.js
```

- Sie entfernen die Daten aus der Repository

```
$ rm test.js
```

- Sie cheken die Daten ein (commiten)

```
$ git commit -m 'alle Javascriptdateien wurden eingechekt'
```

- Sie wechseln zur vorherigen Version

```
$ git checkout [Hashwert des Commits]
```

Branch mit Master zusammenführen (mergen)

Wenn Sie die Arbeit an einem Branch beendet haben und nun dieses Branch mit Masterzweig zusammenführen möchten, dann müssen Sie ins Masterzweig wechseln und von dort aus *merge* ausführen.

```
$ git checkout master
$ git merge myBranch
```

Branch löschen

Sie können nun, wenn Sie möchten, den Branch löschen. Dafür führen Sie den Befehl *branch -d* aus.

```
$ git branch -d myBranch
```

3.4 Repository und Clone zusammenführen (mergen)

Sie sind mit Ihrer Arbeit an der Kopie zufrieden und wollen Ihre Änderungen nun an den entfernten Server übertragen. Der Befehl dafür lautet: *git push [remote-name] [branch-name].*

```
$ git push origin master
```

Wenn Sie den Befehl *git pull* ausfüren, wird der entfernte Branch mit Ihrer Repository zusammengeführt.

---

4. Git-Workflows

Mit Git kann man viele Arbeitsprozesse (workflows) realisieren. In diesem Tutorial wird Ihnen eines davon,Integration-Manager Workflow, erklärt.

Integration-Manager Workflow

Bei diesem Arbeitsablufprozess gibt es ein Hauptrepository, das von einer Person (Integration-Manager) verwaltet wird. Alle Personnen, die am Projekt beteiligt sind, erstellen sich eine Kopie von dem Hauptrepository auf dem Server und arbeiten dann an dieser. Wenn eine Person mit Ihrer Arbeit fertig ist und möchte diese den Anderen zur Verfügung stellen, dann gibt Sie dem Integrations-Manager bescheid. Dieser testen nun die Änderungen und führt Ihre Kopie mit

der Hauptrepository zusammen. Wenn die anderen Personen **git push** ausführen, erhalten Sie Ihre Änderungen. In Abbildung 5 ist dieser Arbeitsablaufprozess zu sehen.

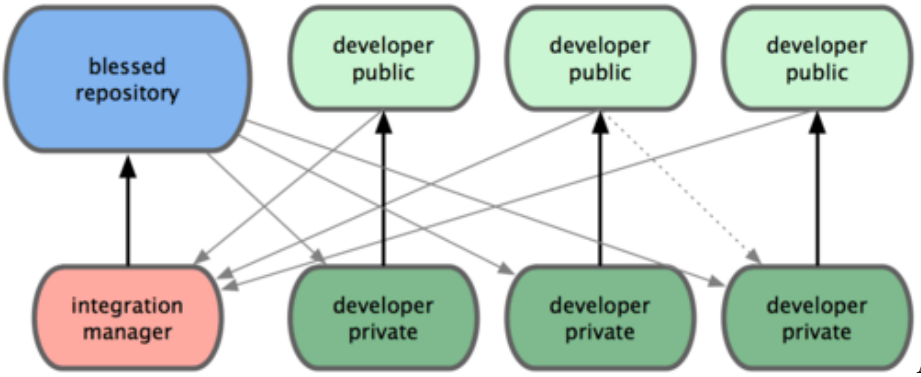

Abbildung 5: Integration-Manager Workflow

. Dies ist ein weit verbreiteter Arbeitsablauf wie ihn z.B. auch GitHub ermöglicht.

---

5. Quellen

- Pro Git: http://git-scm.com/book/de/