git-Tutorial

Eine einfache Anleitung zum Kennenlernen von git

Von Heiderose Borrek

Versionsverwaltung mit git

Git dient zur Verwaltung der verschiedenen Versionen eines Projektes. Es ermöglicht das gleichzeitige Arbeiten an verschiedenen Aspekten des Projekts und das strukturierte Zusammenführen der Ergebnisse. Durch das strukturierte Verwalten der Versionen ist es möglich den Verlauf der Projektbearbeitung und -entwicklung genau nach zu vollziehen.

Die Besonderheit von Git ist die Zweigstruktur des Projektes: Anstatt eines „Master-Branch“ benutzt dieser Workflow zwei Zweige zum Speichern der „Projektgeschichte“ (älteren Versionen).

Master-Branch speichert die offizielle, veröffentlichte „Geschichte“ (die Versionen).

Develop-Branch dient als Integrations-Branch für Funktionen (Features) und Eigenschaften.

Feature-Branch

jedes neue Feature besitzt einen eigenen Feature-Zweig. Dieser kann für ein Backup oder zur Zusammenarbeit in das zentrale Repository geschoben werden. Anstatt des Master-Branch verwendet der Feature-Branch den Develop-Branch als Eltern-Zweig. Ist das Feature komplett, verschmilzt es mit dem Develop-Zweig. Features sollten nie direkt mit dem Master-Branch interagieren.

Release-Branch

Wird vom Develop-Branch abgespalten, wenn genügend neue Features für eine Freigabe vorhanden sind. → Start des Freigabe-Zyklus: Es können keine neuen Features mehr hinzugefügt werden. Nur noch Fehlerbehebungen, Dokumentationen und andere freigabeorientierte Anwendungen. Ist alles soweit fertig bekommt die Freigabeversion eine Versionsnummer und wird mit dem Master-Branch verschmolzen. Zusätzlich sollte der Release-Branch auch zurück in den Develop-Branch gehen, der evtl. schon weiter bearbeitet wurde, seit dem die Freigabe gestartet wurde.

Maintenance-Branch/ Hotfix-Branch

wird benutzt, um schnell Produktionsfreigaben aus zu bessern. Er ist der einzige Branch, der direkt vom Master-Branch abzweigt. Sobald die Lösung komplett ist, sollte der Branch mit dem Master- und Develop-Branch (oder dem aktuellen Release-Branch) verschmelzen und im Master-Branch die neue Version mit einer Versionsnummer ausgezeichnet werden.

In den folgenden Stufen wird die Benutzung von „git“ näher erklärt:  
1. Schritt – Installieren von git  
2. Schritt – Erstellen eines Repositorys  
3. Schritt – Erstellen eines neuen Branchs  
4. Schritt – Veränderungen hinzufügen (add und commit)  
5. Schritt – Verschmelzen von zwei Branches  
6. Schritt – Aktualisieren des Repositorys  
7. Schritt – Pushen des lokalen Repositorys

1. Schritt – Installieren von git

Um mit Git arbeiten zu können müssen Sie sich zunächst git für ihr Betriebssystem (Windows, Linux, OS X…) runter laden und sich bei GitHub (webbasierter Filehosting-Dienst für Software-Entwicklungsprojekte) anmelden.

2. Schritt – Erstellen eines Repositorys

Erstellen Sie ein neues Repository in Git Hub. Um das Repository in Ihr lokales Dateisystem zu kopieren öffnen Sie das Terminal/ die Konsole und springen in das jeweilige Verzeichnis als gewünschten Speicherort. Mit dem Terminalbefehl $ git clone <URL of your repository> kopieren Sie das Repository von GitHub in Ihr lokales Verzeichnis. Nun können Sie mit dem aktuellen Stand weiter arbeiten und später Ihre Ergebnisse wieder veröffentlichen. Beim Klonen des externen Repositorys (Remote Repository – eine Version Ihres Repositorys, die im Internet oder in einem anderen Netzwerk abgespeichert ist) wird automatisch unter dem Namen “origin” ein Verweis darauf angelegt.

Eine Alternative zu der Vorgehensweise oben ist zunächst einen Projektordner lokal zu erstellen, um anschließend das Repository mit dem Befehl $ git init in dem Verzeichnis zu initialisieren. Wenn Sie sich für diese Variante entschieden haben müssen Sie den Master-branch noch selber erstellen, GitHub tut dies schon automatisch für Sie. Auch den Verweis auf das Remote Reposiory müssen Sie mit $ git remote add <name of remote repository> <URL of repository from GitHub> selber erstellen.

Damit Ihre Kollegen ebenfalls auf Grundlage des Repositorys arbeiten können, müssen Sie Ihr lokales Repository auf GitHub veröffentlichen. Dazu pushen Sie das Remote Repository nach GitHub zu dem dort erstellten Pendant (siehe Schritt 7).

Nach dem zweiten Schritt besitzen Sie einen Ordner mit dem aktuellen Stand des Projekts mit dem Sie arbeiten können und ein Repository für dieses Projekt.

3. Schritt – Erstellen eines neuen Branchs

Im dritten Schritt erstellen sie mit $ git checkout -b develop master einen Develop-Branch, der vom Master-Branch abgeht. Von dem Develop-Branch zweigen die Feature- und Release-Branches ab.

Je nachdem an welcher Stelle Sie an dem Projekt arbeiten wollen, erstellen Sie sich mit $ git checkout -b <name of the new branch> <name of the parent-branch> zum Beispiel einen neuen Feature-, Release- oder Hotfix-Branch, wenn dieser noch nicht vorhanden ist. Wollen Sie an einem schon vorhandenen Branch weiter arbeiten können Sie mit $ git checkout <name of the branch> zu diesem Branch springen, um dort z.B. eigene Änderungen zu speichern.

4. Schritt – Änderungen hinzufügen  
add und commit

Entwickelte Änderungen oder neue Funktionen und Eigenschaften des Projekts speichern sie als Commit in dem zugehörigen Branch ab. Dazu speichern Sie die Änderung in dem ab und geben anschließend den Befehl $ git add <name of file> in das Terminal ein. Nun müssen Sie die Änderung noch mit $ git commit -m "note about commit" bestätigen.

Bis zum Beispiel Ihr Feature fertig entwickelt, oder die Release-Prüfung fertig durchgeführt ist speichern Sie die neuen Versionen in dem jeweiligen Branch. Wenn der Teil des Projekts fertig ist kommen Sie zu Schritt fünf.

5. Schritt – Verschmelzen von zwei Branches

Ist zum Beispiel ihr Feature oder Hoffix fertig lassen Sie den Zweig mit dem Develop- bzw. Master-Branch verschmelzen (mergen). Dazu wechseln Sie mit $ git checkout <name of branch> in den Elternzweig mit dem Sie den neuen Branch verschmelzen wollen. Mit

$ git merge <name of new branch> verbinden Sie die zwei Branches. Soll eine Projektversion mit dem Verschmelzen des Release-Branchs mit dem Master-Branch veröffentlicht werden, bekommt die Freigabeversion eine Versionsnummer, die von der Anzahl der Freigaben vor ihr abhängig ist.

Im Moment speichern Sie Ihre Änderungen in ihrem lokalen Repository ab. Möchten Sie ihre Änderungen auch Ihren Kollegen zugänglich machen müssen Sie ihr Repository nach GitHub pushen.

6. Schritt – Aktualisieren des Repositorys

Bevor Sie Ihr lokales Repository nach Github pushen müssen Sie ihr lokales Repository mit den Änderungen, die inzwischen von ihren Kollegen eventuell nach GitHub geladen wurden aktualisieren. Mit dem Befehl $ git fetch <name of the remote> kann man die Daten, die noch nicht auf Ihrem eigenen Rechner verfügbar sind herunterladen. Danach kennt Ihr eigenes Repository Verweise auf alle Zweige und Neuigkeiten, die seit dem Klonen des Repositorys oder dem letzten Aktualisieren hinzu gekommen sind, in dem Remote Repository. Diese können Sie jederzeit mit Ihren eigenen Branches zusammenführen oder durchschauen. Da die Änderungen lediglich runter geladen werden, müssen Sie die Änderungen selbst manuell mit ihren Daten zusammenführen.

7. Schritt – Pushen des lokalen Repositorys

Ist Ihr lokales Repository auf dem Stand des Repositorys von GitHub ergänzt um Ihre Änderungen können Sie Ihr lokales Repository mit $ git push -u <name of the remote repository> <branch-name> nach GitHub pushen.

Zusammenfassung

**Funktion**

**Kommando**

Erstellen eines Repositorys

$ git clone <URL of your repository>

oder

$ git init

Erstellen eines Remote Repositorys

$ git remote add <remote repository> <URL of repository from GitHub>

Erstellen eines neuen Branchs

$ git checkout -b <new branch> <parent-branch>

Zu einem Branch springen

$ git checkout <branch>

Veränderungen hinzufügen

$ git add <file>

$ git commit -m "note about commit"

Verschmelzen von zwei Branches

$ git merge <name of the new branch>

Aktualisieren des Repositorys

$ git fetch <name of the remote repository>

Pushen des lokalen Repositorys

$ git push -u <name of the remote repository> <branch-name>

Begriffe – schnell erklärt

**Branch**

zu deutsch Zweig, ist eine Abspaltung von einer anderen Version, so dass unterschiedliche Versionen parallel weiterentwickelt werden können.

**Repository** Der Speicherort für das Projekt. Alle von der Software verwendeten Daten und Projektversionen werden hier abgelegt.

**Commit** Ausdruck aus der Softwaretechnik, der die bestätigende Freischaltung einer Änderung beschreibt. Vorgang des Einspielens von neuem oder geändertem Quelltext bzw. anderen Dateien in das Versionsverwaltungssystem.

**Git** eine Versionsverwaltung für Dateien, ähnlich SVN und CVS. Der Fokus von Git liegt dabei auf der verteilten Struktur, bei der sich jeder Benutzer eine Kopie des Repositories anlegt, in seiner Kopie Änderungen macht und dann einen sogenannten Pull Request schickt, damit die Änderungen in das Haupt-Repository übernommen werden.

**Revision** Ein Entwicklungszustand einer Software oder allgemein einer Datei.

**Remote Repositorys (Externe Repositorys)** sind Versionen des eigenen Projektes, die im Internet oder irgendwo in einem anderen Netzwerk gespeichert sind. Um mit anderen zusammen arbeiten zu können, muss man solche Repositorys verwalten, Daten aus ihnen herunter- oder heraufladen, um die Arbeit für andere verfügbar zu machen.

**Tag** Wichtige Revision, der ein besonderer Name gegeben wurde, damit man schnell auf sie zugreifen kann.

**Versionsverwaltung** System zur Erfassung von Änderungen an Dokumenten oder Dateien.