GIT Integration für Schüler

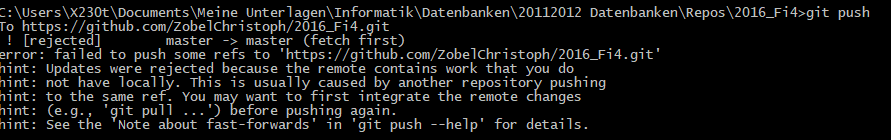
Der erste Entwickler erstellt das Repository

* Der erste Schüler muss auf GIThbu das Repo anlegen
* Dieser Schüler muss auch alle weiteren Gruppenmitglieder im Repo über den Reiter Settings und dort dann Colloberate einladen. Diese müssen hierfür vorab einen kostenfreien Account bei Github anlegen.

Anleitung zum Einrichten des lokalen Repos und Verbinden mit zentralen Repo

* Git herunterladen: <https://git-scm.com/download>
* Ordern auf H: anlegen, der als lokaler Repo-ordner dienen soll
* GIT CMD öffnen
* In den gerate angelegten Ordner in der Konsole navigieren.
* Den Befehl „git clone URL“ ausführen. Alle Dateien von Remote Server müssten nun lokal vorhanden sein.
* Nun kann man Änderungen vornehmen, neue Dokumente anlegen.
* Diese muss man dann „stagen“, d.h. der lokalen Versionsverwaltung mit dem Befehl „git add .“ zuordnen.
* Anschließend sind diese zu commiten, mit dem Befehl „git commit –m „Text des Commits“
* Zuletzt sind die Änderungen an das Remote Repository mit dem Befehl „git push -u origin master“ zu pushen.

Mergen bei Konflikten



Sofern zwei Entwickler an unterschiedlichen Versionen einer Datei gearbeitet haben und der erste Entwickler seine Änderungen bereits an das Remote Repository übermittelt hat, muss der zweite Entwickler vor dem Pushen die Versionen mergen.

* Dies geht mit dem Befehl „git fetch origin“, der die aktuelle Version vom Server herunterlädt.
* Der Befehl „git merge“ führt dann die Versionen zusammen. Falls es Mergekonflikte gibt, sind diese zusätzlich noch zu beheben – hier ist das Dokument zu Bearbeiten und danschließend zu stagen(git add .) und zudem zu commiten (git commit –m „Text“) - siehe in diesem Fall nachfolgende Quellen.
* Anschließend kann die zusammengeführte Version gepushed werden, dies mit dem bekannten Befehl „git push“.

Empfohlene Quellen für weitere Informationen:

* <https://rogerdudler.github.io/git-guide/index.de.html>
* <https://git-scm.com/book/de/v1/Distribuierte-Arbeit-mit-Git-xxx-An-einem-Projekt-mitarbeiten>

Nachfolgend ist das Arbeiten in kleinen Teams beschrieben. Quelle: <https://git-scm.com/book/de/v1/Distribuierte-Arbeit-mit-Git-xxx-An-einem-Projekt-mitarbeiten>

### [Kleine Teams](https://git-scm.com/book/de/v1/Distribuierte-Arbeit-mit-Git-xxx-An-einem-Projekt-mitarbeiten#Kleine-Teams)

Das einfachste Setup, mit dem Du zu tun haben wirst, ist ein privates Projekt mit ein oder zwei Entwicklern. Mit „privat“ meine ich, dass es „closed source“, d.h. nicht lesbar für Dritte ist. Alle beteiligten Entwickler haben Schreibzugriff auf das Repository.

In einer solchen Umgebung kann man einen ähnlichen Workflow verwenden, wie für Subversion oder ein anderes zentralisiertes System. Du hast dann immer noch Vorteile wie, dass Du offline committen kannst und dass Branching und Merging so unglaublich einfach ist. Der Hauptunterschied ist, dass Merges auf der Client Seite stattfinden und nicht, wenn man committet, auf dem Server. Schauen wir uns an, wie die Arbeit von zwei Entwicklern in einem gemeinsamen Repository abläuft. Der erste Entwickler, John, klont das Repository, nimmt eine Änderung vor und comittet auf seinem Rechner. (Wir kürzen die Beispiele etwas ab und ersetzen die hierfür irrelevanten Protokoll Meldungen mit xxx.)

# John's Machine

$ git clone john@githost:simplegit.git

Initialized empty Git repository in /home/john/simplegit/.git/

...

$ cd simplegit/

$ vim lib/simplegit.rb

$ git commit -am 'removed invalid default value'

[master 738ee87] removed invalid default value

1 files changed, 1 insertions(+), 1 deletions(-)

Der zweite Entwickler, Jessica, tut das gleiche. Sie klont das Repository und committet eine Änderung:

# Jessica's Machine

$ git clone jessica@githost:simplegit.git

Initialized empty Git repository in /home/jessica/simplegit/.git/

...

$ cd simplegit/

$ vim TODO

$ git commit -am 'add reset task'

[master fbff5bc] add reset task

1 files changed, 1 insertions(+), 0 deletions(-)

Jetzt lädt Jessica ihre Arbeit mit git push auf den Server:

# Jessica's Machine

$ git push origin master

...

To jessica@githost:simplegit.git

1edee6b..fbff5bc master -> master

John versucht, das selbe zu tun:

# John's Machine

$ git push origin master

To john@githost:simplegit.git

! [rejected] master -> master (non-fast forward)

error: failed to push some refs to 'john@githost:simplegit.git'

John darf seine Änderung nicht pushen, weil Jessica in der Zwischenzeit gepushed hat. Dies ist ein Unterschied zu Subversion: wie Du siehst, haben die beiden Entwickler nicht dieselbe Datei bearbeitet. Während Subversion automatisch merged, wenn lediglich verschiedene Dateien bearbeitet wurden, muss man Commits in Git lokal mergen. John muss also Jessicas Änderungen herunterladen und mergen, bevor er dann selbst pushen darf:

$ git fetch origin

...

From john@githost:simplegit

+ 049d078...fbff5bc master -> origin/master

Zu diesem Zeitpunkt sieht Johns lokales Repository jetzt aus wie in Bild 5-4.

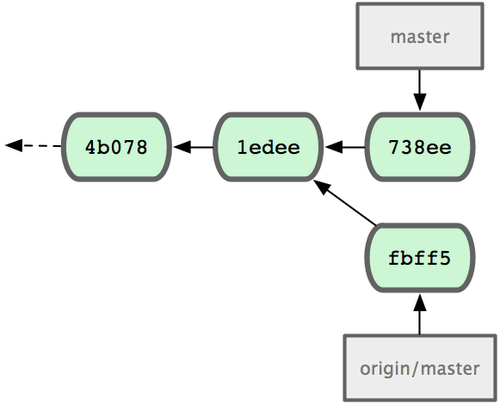


Bild 5-4. Johns ursprüngliches Repository

John hat eine Referenz auf Jessicas Änderungen, aber er muss sie mit seinen eigenen Änderungen mergen, bevor er auf den Server pushen darf:

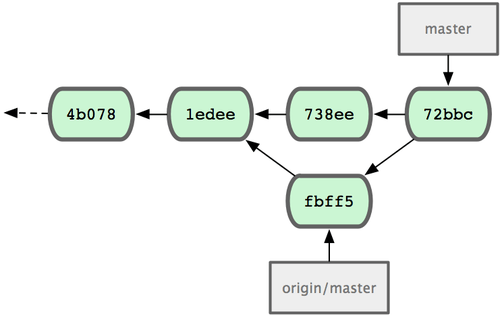
$ git merge origin/master

Merge made by recursive.

TODO | 1 +

1 files changed, 1 insertions(+), 0 deletions(-)

Der Merge verläuft glatt: Johns Commit Historie sieht jetzt aus wie in Bild 5-5.



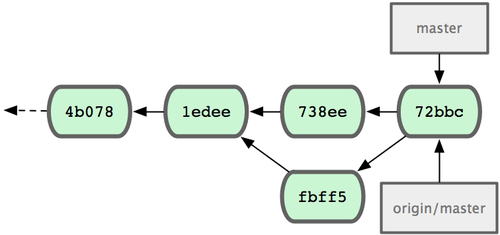
Johns Repository nach dem Merge mit origin/master

John sollte seinen Code jetzt testen, um sicher zu stellen, dass alles weiterhin funktioniert. Dann kann er seine Arbeit auf den Server pushen:

$ git push origin master

...

To john@githost:simplegit.git

 fbff5bc..72bbc59 master -> master

Johns Commit Historie sieht schließlich aus wie in Bild 5-6.

Johns Commit Historie nach dem pushen auf den origin Server

In der Zwischenzeit hat Jessica auf einem Topic Branch (xxx) gearbeitet. Sie hat einen Topic Branch mit dem Namen issue54 und darin drei Commits angelegt. Sie hat Johns Änderungen bisher noch nicht herunter geladen, sodass ihre Commit Historie jetzt so aussieht wie in Bild 5-7.

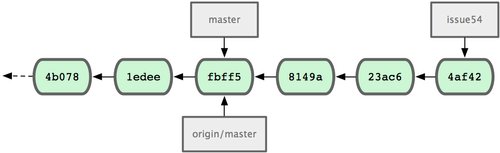


Bild 5-7. Jessicas ursprüngliche Commit Historie

Jessica will ihre Arbeit jetzt mit John synchronisieren. Also lädt sie seine Änderungen herunter:

# Jessica's Machine

$ git fetch origin

...

From jessica@githost:simplegit

fbff5bc..72bbc59 master -> origin/master

Das lädt die Änderungen, die John in der Zwischenzeit hochgeladen hat. Jessicas Historie entspricht jetzt Bild 5-8.

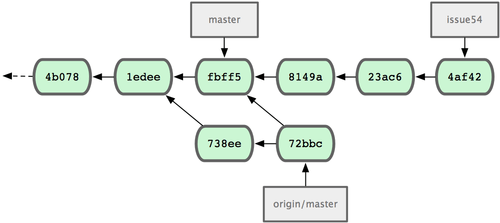


Bild 5-8. Jessicas Historie nachdem sie Johns Änderungen geladen hat

Jessica hat die Arbeit in ihrem Topic Branch abgeschlossen, aber sie will wissen, welche neuen Änderungen es gibt, mit denen sie ihre eigenen mergen muss.

$ git log --no-merges origin/master ^issue54

commit 738ee872852dfaa9d6634e0dea7a324040193016

Author: John Smith <jsmith@example.com>

Date: Fri May 29 16:01:27 2009 -0700

removed invalid default value

Jetzt kann Jessica zunächst ihren Topic Branch issue54 in ihren master Branch mergen, dann Johns Änderungen aus origin/master in ihren master Branch mergen und schließlich das Resultat auf den origin Server pushen. Als erstes wechselt sie zurück auf ihren master Branch:

$ git checkout master

Switched to branch "master"

Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.

Sie kann jetzt entweder origin/master oder issue54 zuerst mergen – sie sind beide „upstream“ (xxx). Der resultierende Snapshot wäre identisch, egal in welcher Reihenfolge sie beide Branches in ihren master Branch merged, lediglich die Historie würde natürlich minimal anders aussehen. Jessica entscheidet sich, issue54 zuerst zu mergen:

$ git merge issue54

Updating fbff5bc..4af4298

Fast forward

README | 1 +

lib/simplegit.rb | 6 +++++-

2 files changed, 6 insertions(+), 1 deletions(-)

Das ging glatt, wie Du siehst, war es ein einfacher „fast-forward“ Merge. Als nächstes merged Jessica Johns Änderungen aus origin/master:

$ git merge origin/master

Auto-merging lib/simplegit.rb

Merge made by recursive.

lib/simplegit.rb | 2 +-

1 files changed, 1 insertions(+), 1 deletions(-)

Auch hier treten keine Konflikte auf. Jessicas Historie sieht jetzt wie folgt aus (Bild 5-9).

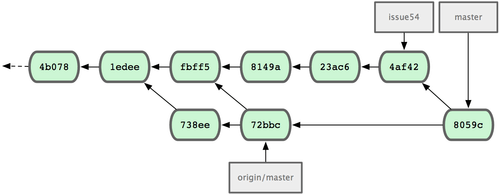


Bild 5-9. Jessicas Historie nach dem Merge mit Johns Änderungen

origin/master ist jetzt in Jessicas master Branch enthalten (xxx reachable xxx), sodass sie in der Lage sein sollte, auf den origin Server zu pushen (vorausgesetzt, John hat zwischenzeitlich nicht gepusht):

$ git push origin master

...

To jessica@githost:simplegit.git

72bbc59..8059c15 master -> master

Beide Entwickler haben jetzt einige Male committed und die Arbeit des jeweils anderen erfolgreich mit ihrer eigenen zusammengeführt.

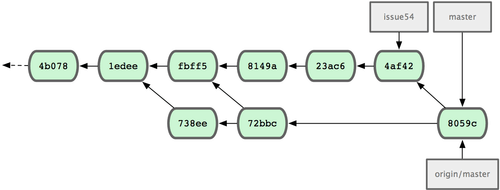


Bild 5-10. Jessicas Historie nachdem sie sämtliche Änderungen auf den Server gepusht hat

Dies ist eine der simpelsten Workflow Varianten. Du arbeitest eine Weile, normalerweise in einem Topic Branch, und mergst in Deinen master Branch, wenn Du fertig bist. Wenn Du Deine Änderungen anderen zur Verfügung stellen willst, holst Du den aktuellen origin/master Branch, mergst Deinen master Branch damit und pushst das ganze zurück auf den origin Server. Der Ablauf sieht in etwa wie folgt aus (Bild 5-11).

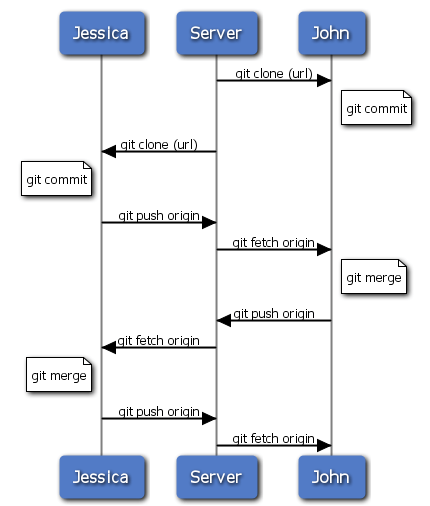


Bild 5-11. Ablauf eines einfachen Workflows für mehrere Entwickler