Git & Github

Eine der Stärken von Git ist, dass es die Zusammenarbeit von EntwicklerInnen unterstützt: Sie können im großen Stil an Projekten arbeiten und geordnet Änderungen austauschen.

Git ist ein verteiltes Versionssystem, d. h. es gibt typischerweise nicht nur ein lokales Repository bei Euch sondern weitere *remote*-Repositories auf anderen Rechnern, mit denen Änderungen ausgetauscht werden können. Zwar wollen wir zunächst nur jeder in seinem lokalen Repository arbeiten, doch soll es für unsere späteren Experimente gleich mit einem remote-Repository verbunden sein.[1]

Github ist das *Social Network* für Entwickler. Es verwaltet öffentlich zugängliche (und private) Repositories mit Git und bietet darüber hinaus zahlreiche Funktionen zur Projektverwaltung und Präsentation des Projekts im Netz.

`git status` ist Euer Freund. Wann immer Ihr in Git verloren seid, könnt ihr damit sehen, wie Git die Lage beurteilt. Seht Euch die Ausgabe an und versucht zu verstehen, was Git Euch damit sagen will.

Repositories anlegen: Clonen

Die einfachste Art, eine Verbindung zwischen Repositories herzustellen, ist ein neues Repository als vollständige Kopie (inklusive der gesamten Historie) eines bestehenden Repositories anzufertigen. Das geschieht mit `git clone` Repository-URL. Der Einfachheit halber nehmen wir ein schon bestehendes Repository bei Github:

1. Bitte kopiert Euch das Repository `git-workshop` unter `github.com/fh-wedel`:
   `git clone https://github.com/fh-wedel/git-workshop.git`

   Damit gibt es bei Euch nun ein lokales Repository im Verzeichnis `git-workshop`, das Ihr nun bearbeiten könnt. Seht Euch an, welche Files Git zur Verwaltung erzeugt (Ihr müsst sie nicht im Detail verstehen, aber wo sie stehen ist schon interessant.)

1 Jeder für sich: Arbeiten im lokalen Repository

Unser hier zu entwickelndes Mini-Projekt soll ein Web-Server für den Git-Workshop selbst sein. Dafür brauchen wir auch ein paar Web-Seiten in HTML. Die findet man in `src/html`.

Im aktuellen Branch arbeiten

Ohne weitere Maßnahmen arbeitet Ihr auf dem `master`–Branch. Es ist in Git üblich, mit eigenen Branches zu arbeiten und nicht direkt auf `master`. Wir machen das jetzt trotzdem erstmal.

2. Legt bitte ein Unterverzeichnis mit Eurem Namen in `src/html` an, erstellt dort bitte ein paar HTML-Files und bearbeitet sie so, wie Ihr wollt. Was sagt `git status`? Nehmt

[1] Ein isoliertes lokales Repository kann man ganz in einem Verzeichnis eigener Wahl einfach mit `git init` erzeugen. Machen wir hier aber nicht.

(einige) Files für Euren ersten Commit auf (das sog. *stagen* mit `git add filename`, was sagt `git status`?) und macht den Commit (`git commit -m"Commit-Meldung"`). Was sagt `git status`?

Weitere nützliche Kommandos, um zu sehen, wie Eure Änderungen aussehen sind `git diff` (zeigt den Unterschied zwischen lokalen Änderungen und gestageten Files) und `git show` (zeigt die Änderungen des letzen Commits).

Branchen

Git macht es leicht, für einzelne Aufgaben Branches anzulegen, in denen die Arbeiten für diese Aufgabe zusammengefasst werden. Es ist üblicher Git-Stil, dass man für jede Teilaufgabe einen Branch anlegt, darin die notwendigen Änderungen macht und den Branch dann zurück in den Haupt-Branch merget. So kann man die Änderungen für einzelne Features voneinander trennen und auch gezielt und ausgewählt zur Verfügung stellen.

3. Macht wieder einige Änderungen in Euren lokalen Files (noch kein Stagen oder Committen). Vielleicht fügt Ihr `style`-Angaben hinzu oder erzeugt sogar ein eigenes Stylesheet.

   Nachdem Ihr Eure Änderungen gemacht habt, stellt Ihr fest, dass sie besser auf einem Branch aufgehoben sind (vielleicht ist die Aufgabe, einige Passagen in rot hervorzuheben). Mit `git checkout -b Branch-Name` könnt Ihr nachträglich den Branch erzeugen, in dem Ihr dann Eure Änderungen stagen und commiten könnt. Macht mal einen Branch und nennt Ihn einfach `texte-hervorheben`.

   Was sagt eigentlich `git status`?

4. Arbeitet so auf Eurem neuen Branch und macht ein paar Commits.

Vielleicht stellt Ihr fest, dass die Arbeiten auf dem Branch in eine Sackgasse geführt haben und Ihr sie gar nicht weiterverwenden wollt. Dann kann man einfach den Branch Branch sein lassen und zurück auf `master` wechseln: `git checkout master` (natürlich kann man den Branch auch wieder löschen, muss man aber nicht).

Aber meist, wollt Ihr die Änderungen im Branch weiterverwenden. Sie werden dafür ge*merge*t.

Mergen

Das Mergen erfolgt bei Git immer in einen Branch hinein, d. h. man wechselt also erst in den Branch in den man mergen will und löst dann das Mergen aus.

5. Übernehmt Eure Änderungen von Eurem Branch auf den `master`-Branch (`git checkout master; git merge --no-ff Branch-Name`[2]).

   Wenn es keine konkurrierenden Änderungen auf dem `master`–Branch gibt, dann sollte das Mergen ohne Probleme erfolgen und auch gleich commited werden.

   Gibt es konkurrierende Änderungen, entstehen möglicherweise Konflikte. Git committed den Merge nicht, Ihr müsst die Konflikte sinnvollerweise beheben und dann selbst (stagen und) committen.

[2] `--no-ff` bedeutet, dass bestimmt kein sog. *Fast-Forward-Merge* gemacht wird, bei dem nicht wirklich gemerget wird, sondern nur Referenzen verschoben werden.

2 Alle miteinander: Arbeiten mit entferntem Repository

Git-Repositories können (und sind oft) mit anderen Repositories (sog. *remotes*) verbunden. Zwischen diesen Repositories können Commits ausgetauscht werden.

Wir hatten unser Repository ja durch `git clone` erzeugt und dabei wird gleich eine Verbindung zum Ursprungs-Repository unter dem Namen `origin` hergestellt. Das kann man mit `git remote -v` sehen.

Pull und Push

Mit `git pull` werden Commits aus einem Remote-Repository in das lokale Repository übernommen. Mit `git push` werden Commits aus dem lokalen Repository in ein Remote-Repository übertragen.

6. Macht bitte wie zuvor Änderungen in Eurem lokalen Repository (Eigener Branch, Änderungen, commit, Änderungen, commit, mergen) und pusht dann Eure Änderungen in das Github-`git-workshop`-Repository. Werden Eure lokalen Branches und die Commits darauf auch mitübertragen?

7. Bitte passt das File `src/html/index.html` an und tragt Euch in die Teilnehmerliste (gerne mit Pseudonym) ein. Stellt auch diese Eure Änderung bereit.

3 Github: Fork und Pull Request

Bisher habt Ihr mit einem Repository gearbeitet, für das Ihr Schreibzugriff hattet. Github ermöglich es aber auch, Änderungen an einem schreibgeschützten Repository an den Autor zu senden.

Fork

Auf der Github-Seite des Repositories drückt Ihr rechts oben den Knopf `Fork`. Damit erstellt Ihr eure eigene Kopie des Repositories, auf die Ihr auch schreibend zugreifen könnt.

Diese Kopie klont Ihr bitte auf Euren lokalen Rechner, nehmt einige Änderungen vor, und pushed sie auf Github. Dabei ist zu beachten, daß `fetch`, `pull` und `push` sich auf Euren Klon beziehen. Es ist aber auch möglich, auf das ursprüngliche Repository zuzugreifen, indem mit `git remote add upstream https://github.com/otheruser/repo.git` ein zusätzliches remote hinzugefügt wird. Damit könnt Ihr während Eurer Arbeit Änderungen des Upstream integrieren (z.B. um sicherzustellen, daß Eure Änderungen zu der jeweils aktuellen Version passen).

Pull Request

Um dem Besitzer des ursprünglichen Repositories die Änderungen zukommen zu lassen, erstellt Ihr einen `Pull Request`, den Knopf findet Ihr ebenfalls rechts oben.

Der Pull Request enthält alle Änderungen Eures geforkten Repositories; auch die, die Ihr nach dem Anlegen des Pull Request hinzufügt. Dadurch könnt Ihr nachträglich Änderungen zum Pull Request hinzufügen und so z.B. auf Änderungsanforderungen reagieren oder später entdeckte Fehler korrigieren.

Änderungen und Fehler werden bei Github direkt im Pull Request bzw. im Commit diskutiert. Neben den klassischen Kommentaren lassen sich mit dem blauen Plus, daß neben der jeweils aktiven Zeile erscheint, Diffs inline kommentieren. Probiert das am besten einfach mal untereinander aus. Kommentierte Commits erhalten ein zusätzliches Icon in der Liste aller Commits, lassen sich also leicht wiederfinden, Pull Requests bieten eine Liste aller Kommentare.

Git Revisions

Wie adressiere ich eine Revision?

- Mit dem SHA-Wert, der im Log anzeigt wird. Dieser kann von hinten abgeschnitten werden, so lange der Bezeichner im Repository eindeutig ist.
- Mit der Ausgabe von `git describe`.
- Mit einer symbolischen Referenz, die in im Repository definiert ist, z.B. `master`, `HEAD`, `FETCH_HEAD`, `ORIG_HEAD`
- Mit einer Kombination aus Referenz und Datum: `master@{yesterday}`, `HEAD@{5 minutes ago}`. Diese Adressen liefern den Zustand zu dem spezifizierten Zeitpunkt, um Änderungen innerhalb eines Zeitraumes anzuzeigen eignen sich `--since` und `--until`.
- Vorläufer einer Referenz: `HEAD 10`
- Eine Reihe von Commits wird genau wie ein einzelnes Commit adressiert. Befehle, die eine Reihe erwarten gehen dann Kette der Vorfahren durch, um die vollständige Liste zu erzeugen, z.B. zeigt `git log 73a34` keine Commits nach 73a34 an.
- Um Commits aus einer Reihe auszuschließen, wird die Notation `^` verwendet, `^r1 r2` meint alle Commits, die von r2, nicht aber von r1 aus erreichbar sind. Beispielsweise zeigt `git log ^73a34 HEAD` die Commits von dem Nachfolger von `73a34` bis `HEAD` an.
- r1..r2 bezeichnet alle Commits, die von r2, nicht aber von r1 zu erreichen sind. Die ähnliche Notation r1...r2 bezeichnet alle Commits, die entweder von r1 oder von r2, aber nicht von beiden aus zu erreichen sind.

Nützliche Links

Git Quick reference
`http://jonas.nitro.dk/git/quick-reference.html`

Weitere Fähigkeiten von Git

bisect

Suche nach einem fehlerhaftem in einer Reihe von Commits. Mittels `git bisect start` wird bisect gestartet, mittels `git bisect bad` und `git bisect good` werden defekte und heile Fassungen markiert. Bisect teilt die Commits innerhalb der Markierungen auf und checkt eine Revision in der Mitte aus, die getestet und dann entsprechend markiert werden muß. Zum Schluß bleibt die defekte Revision übrig.

Die Hilfe (`git help bisect`) bietet detaillierte Beschreibungen zu einer Reihe weiterer Befehle, z.B. zum Zurücksetzen des Zustandes, Betrachten der Historie, Überspringen von Commits etc.

bundle

Archiv-Datei aus einem Git-Repository erstellen. Komprimierte Bundle-Dateien stellen die kleinste Repräsentation eines Repositories dar. Daher eignen sie sich hervorragend für den Versand per E-Mail oder Transport auf einem USB-Stick.

Mit `git bundle create repo.bundle master` werden alle zum Branch `master` gehörenden commits in die Datei `repo.bundle` geschrieben. Es ist auch möglich, kleinere bundles zu erzeugen, z.B. mit `--since=10.days master` oder `v2.0..master`.

Mit `clone`, `fetch` und `pull` werden die Dateien eingelesen, z.B. `git clone repo.bundle newrepo`. Weitere Befehle wie `verify` und `unbundle` werden in der Hilfe beschrieben.

cherry-pick

Einzelne Comics übernehmen. Anstelle eines vollständigen merge ist es mit cherry-pick möglich, einzelne Commits aus einem Branch zu übernehmen. Dies funktioniert in der Regel auch dann, wenn sich die beteiligten Branches sehr weit voneinander weg entwickelt haben und beispielsweise das Ziel des ursprünglichen Commits inzwischen in einer anderen Datei liegt.

Beispiel: `git cherry-pick master` übernimmt die Änderungen des tip vom Branch master und erzeugt einen neuen Commit. `git cherry-pick maint master..next` übernimmt alle Commits, die Vorgänger von maint oder next, nicht aber von master sind.

grep

Dateien und Daten im Repository durchsuchen. `git grep "ElasticSearchNullPointerException"` durchsucht lokale Dateien, `git grep --cached "ElasticSearchNullPointerException"` durchsucht die im Index registrierten Blogs und `git grep "ElasticSearchNullPointerException" 575250~50` durchsucht das 50. Commit vor dem Commit `575250e223c31218b13e927e914925299594973d`.

stash

Änderungen temporär speichern, um ein sauberes Arbeistverzeichnis zu erhalten, ohne Arbeit zu verlieren. `git stash save` speichert den aktuellen Stand, `git stash list` zeigt eine Liste aller gespeicherten Zwischenstände an und `git stash pop` übernimmt die gespeicherten Änderungen ins Arbeitsverzeichnis, ohne die Änderungen zum Index hinzuzufügen oder zu comitten.

svn

Ein bidirektionaler Subversion¡=¿Git Adapter. Ein Subversion-Repository wird per `git`

`svn clone https:/...` geklont (das lokale Git-Repostory enthält danach sämtliche Commits aus Subversion); das daraus entstandene Repository kann wie jedes andere Git-Repository verwendet werden. Commits können per `git svn dcommit` in das Subversion-Repository zurück übertragen und Änderungen aus dem Subversion-Repository per `git svn rebase` geladen werden.

Bei der Interaktion mit Branches ist Vorsicht geboten, aufgrund des Umfanges des sehr mächtigen Subversion-Adapters wird empfohlen vor Verwendung die Hilfseite (`git help svn`) zu lesen.

submodule

Repositories schachteln: Mittels `git submodule add git://... lib/extern1` wird ein Git-Repostory unterhalb des Verzeichnisses `lib/extern1` eingebunden, mit `git submodule init` initialisiert und mit `git submodule update` ausgecheckt.

Das Commit des Submodules wird im übergeordneten Repository gespeichert, so daß ein erneutes `git submodule update` immer die gleiche Version auscheckt. Um das Submodul zu aktualisieren, muß zuerst das Submodul aktualisiert werden, z.B. mit `cd lib/extern1; git pull origin master` und dann mittels `cd ..; git add extern1; git commit...` im übergeordneten Repository gespeichert werden.

notes

Wenn ein Commit nicht hinreichend Kommentiert ist, aber auch nicht mehr geändert werden soll, so kann mittels `git note add -m "..." <rev>` eine Notiz hinzugefügt werden.

archive

Mit `git archive --prefix=xyz-1.4.0/ -o xyz-1.4.0.tar.gz v1.4.0` wird ein Archiv der Daten ohne Git-Metainformationen erstellt (entspricht `svn export`).

reflog

`git reflog` zeigt die letzten Aktionen auf dem Repository an, in der Voreinstellung werden die letzten 90 Tage gespeichert. Mit diesen Informationen ist es in vielen Fällen möglich, verloren gegangene Commits (z.B. durch ein voreiliges `git reset --hard HEAD^`) wiederherzustellen.

Um Platz zu sparen können reflog-Einträge gelöscht werden, z.B. so: `git reflog expire --expire=1.minute refs/heads/master; git fsck --unreachable; git prune; git gc`.