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.

Änderungen am Projekt werden dabei immer zunächst in einem lokalen Repository vorgenommen — was wir am Anfang üben wollen — und erst dann, wenn die Änderungen erfolgreich sind, anderen in einem globalen Repository[1] zur Verfügung gestellt. Das sehen wir uns im Anschluss an.

Github ist das *Social Network* für Entwickler. Es verwaltet öffentlich zugängliche (und private) Repositories, erlaubt ihre Verwaltung 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.

1 Jeder für sich: Arbeiten im lokalen Repository

Repositories anlegen

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.

1. Legt bitte ein Arbeitsverzeichnis an, in dem Ihr Eure HTML-Files verwalten wollt. Dieses Verzeichnis macht Ihr mit `git init` zu einem Git-Repository. Seht Euch an, welche Files Git zur Verwaltung erzeugt (Ihr müsst sie nicht im Detail verstehen, aber wo sie stehen ist schon interessant.)

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 HTML-Files in Eurem Arbeitsverzeichnis an und bearbeitet sie so, wie Ihr wollt. Was sagt `git status`? Nehmt (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).

[1] Technisch weisen lokale und globale Repositories die gleiche Struktur auf. Es geht hier also um Konventionen der Verwendung und nicht um technische Eigenschaften der Repositories.

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 merged. 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

Dass 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 behebe, 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

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.

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.

6. 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`, dass Ihr wie bisher bearbeiten könnt.

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.

7. 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?

Nützliche Links

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