Git und Gitlab

Effiziente Bereitstellung wesentlicher und stets aktueller Projektinformationen sowie Konfigurationsmanagement sind entscheidende Faktoren für erfolgreiche Entwicklungsprojekte – und zwar von Beginn an!

Konfigurationsmanagement ist eine der zentralen Aufgaben im Software-Engineering. Erst damit wird eine vernünftige Softwareentwicklung im Team möglich. Zwei der bekanntesten freien Werkzeuge sind ***SVN*** (Subversion) und ***Git***. Mit diesen Werkzeugen ist es möglich, Code (also ASCII-Texte) und binäre Dateien (Worddokumente) professionell zu verwalten.
Im Rahmen dieser Übung beschäftigen wir uns nun mit grundlegenden Git Kommandos.

Für die Darstellung aktueller Projektinformationen sind Wiki Seiten ein einfaches und probates Mittel. Auch hier gibt es wieder verschiedenste integrierte Spielarten (*Redmine, GitLab, trac*, etc.). In dieser Übung sehen wir uns *GitLab* an.

Ein Projekt benötigt aber zunächst einmal ein so genanntes Repository …

Aufgabe 1 – Vorarbeiten: Teams und Teamserver Repositories

a) Bilden Sie Teams mit höchstens **fünf** Personen und bestimmen Sie je einen Projektleiter für das Projekt-Repository.
b) Der Teamadministrator legt ein Projekt auf dem GitLab[1] an.
   - Projektname: prg3-inf-txx (xx = Teamnummer)
   - Als Login verwenden Sie ihre RZ-Kennung und Passwort (also wie beim Login auf der FH-Rosenheim Website)
   - Auf der Loginseite rechts oben einen Button „**New project**“, klicken und für das Team den zugehörigen Namen bei „**Projekt Path**“ eingeben. Zudem eine passende Beschreibung für das Projekt – bei „**Visibility Level**“ ist hierbei „**Private**“ auszuwählen. Das somit erstellte Projekt wird auf der Übersichtsseite angezeigt.

Aufgabe 2 – Teamadministration

Nach dem Erstellen des Projekts muss der Eigentümer des Projekts (Teamadministrator) nun die Teammitglieder hinzufügen. Damit dies möglich ist, müssen diese sich zumindest einmal am Gitlab angemeldet haben.
Wenn dies geschehen ist, können im Projekt unter „**Members**“ Teammitglieder hinzugefügt werden. Die Suche berücksichtigt sowohl die RZ-Kennungen (Anzeigename) als auch FH-Emailadresse sowie Vor- und Nachnamen.

a) Weisen Sie den Teammitgliedern die notwendigen Nutzungsrechte zu („**Project Access**“).
b) Prüfen Sie (jedes Teammitglied), ob Sie Zugang zu Ihrem Projekt haben.

[1] https://inf-git.fh-rosenheim.de/

Aufgabe 3 – Erstellung Projekt Homepage (Wiki)

Es besteht die Möglichkeit, ein Projektwiki anzulegen:

- https://inf-git.fh-rosenheim.de/help/markdown/markdown

Aufgabe 4 – Anlegen und Clonen des Repositories

Jede Person Ihres Teams hat eine vollständige Version aller Dateien des Projekts in einem lokalen Verzeichnis. Über Git wird der Inhalt des lokalen Verzeichnisses mit dem zentralen (remote) Repository abgeglichen.
Mit dem Anlegen des Projekts wird automatisch ein Repository auf dem Server erzeugt. Um nun ihre Daten mit dem Repository zu synchronisieren, benötigen Sie ein passendes Tool. Eine Liste von Alternativen zu SourceTree (Git GUI) finden Sie hier[2]. In jedem Fall allerdings zunächst Git für ihre Plattform herunterladen und installieren:

- Windows: https://git-scm.com/download/win
- Linux: https://git-scm.com/download/linux
- Mac: https://git-scm.com/download/mac

Anschließend wird die Benutzung von mit SourceTree beschrieben, die Verwendung der anderen Tools oder auch der Kommandozeile ist ohne weiteres möglich.
Ein Tutorial für die Verwendung des Kommandozeilentools finden Sie hier[3].

Aufgabe 4.1 – Authentifizierung via Benutzername und Passwort

Der schnellste Weg ist die Authentifizierung über Benutzername und Passwort. Hierfür wählen Sie in SourcTree „**Klonen / Neu“** aus, und fügen Sie die im Gitlab-Projekt angegebene „**https:** *“* URL an, also beispielsweise:

https://inf-git.fh-rosenheim.de/sINFmamus/se1-inf-txx.git

In dem von Ihnen ausgewählten Verzeichnis (Zielpfad) können Sie nun Dateien anlegen und diese im zentralen Repository abspeichern.

Die Verwendung der Authentifizierung via SSH-Key ist in den Zusatzaufgaben am Ende des Aufgabenblatts beschrieben.

Der Unterschied zwischen Klonen und Pullen ist, dass der Klon zur Initialisierung des Workspace und Lokalen Repositories verwendet wird, wohingegen ein Pull nur Änderungen aus dem Repository holt.

Das Repository muss nicht händisch angelegt werden, da durch das Erstellen des Projekts dies bereits erledigt wurde.

[2] https://git-scm.com/downloads/guis
[3] https://try.github.io/levels/1/challenges/1

Aufgabe 5 – Dateien dem Team zur Verfügung stellen

Ein Teammitglied erzeugt nun im Projektverzeichnis ein Unterverzeichnis „`source`“ und darin eine Datei „`.gitkeep`“ (gitkeep[4]).

- Unter Windows muss hierbei ein zusätzlicher Punkt am Ende eingefügt werden also [Punkt]gitkeep[Punkt] und in den Ordneroptionen der Haken unter „Ansicht“ -> „Erweiterungen bei bekannten Dateitypen ausblenden“ entfernt werden.

Die Gitkeep-Datei wird nur zum Anlegen von Ordnerstrukturen benötigt und kann wieder entfernt werden, sobald die Ordner mit Daten befüllt sind, da leere Ordner prinzipiell nicht versionstechnisch erfasst sind.

Zusätzlich soll im Root-Verzeichnis noch eine Datei „`HelloWorld.txt`“ angelegt werden. Diese Datei kann nach erstellen in SourceTree den anderen Teammitgliedern zur Verfügung gestellt werden. Hierfür in SourceTree das Projekt-Repository auswählen, und entweder in der Menüleiste auf „**Hinzufügen**“, oder bei „**Nicht vorgemerkte Dateien**“ den Haken der Checkbox (neben dem Verzeichnis und der Datei) setzen und auf „**Commit**“ klicken.

> Es ist guter Stil, bei jedem **Commit** einen **sinnvollen Kommentar** anzugeben – hier zum Beispiel „Initiale Erstellung“.

Mit „**Commit**“ werden die Änderungen in das lokalen Repository übertragen. Um diese Änderungen nun auf das Remote Repository zu übertragen, und damit den Teammitgliedern den Zugriff zu ermöglichen, muss ein „**Push**“ durchgeführt werden. Bei dem Push-Dialog klicken Sie auf die Checkbox neben „**master**“

Alle Teammitglieder Klonen sich nun das Projekt wie oben beschrieben. Falls schon geschehen, einen „**Pull**“ durchführen. Danach sollte die erzeugte Datei „`HelloWorld.txt`“ zusammen mit dem Ordner „`source`“ und der .gitkeep-Datei für alle anderen Teammitglieder sichtbar sein.

Aufgabe 5.1 – Einarbeitung in die Funktionsweise von GIT

Neben der .gitkeep gibt es noch die Möglichkeit .gitignore-Dateien im Repository zu hinterlegen. So wird Git mitgeteilt, welche Dateien ignoriert werden sollen. Dies kann separat für jedes Verzeichnis angegeben werden – siehe hier[5]. Erstellen Sie im Ordner „source“ eine .gitignore Datei mit dem Inhalt „*.tmp“ und fügen Sie zusätzlich eine Datei mit der Endung .tmp ein.

Beobachten sie das Verhalten von SourceTree bei Ihrem nächsten Commit.

Vollziehen Sie den Ablauf von:

1) Pull
2) Durchführung von Lokal Änderungen
3) Hinzufügen von nicht vorgemerkten Dateien,
4) Commit, dann Schritt 2-4 beliebig oft wiederholen
5) Pull
6) Push

---

[4] http://stackoverflow.com/questions/7229885/what-are-the-differences-between-gitignore-and-gitkeep/7229996#7229996

[5] https://git-scm.com/docs/gitignore

unter Zuhilfenahme des Cheatsheets [6]nach. Der Ablauf 1-6 sollte immer die typische Vorgehensweise sein. Es ist wichtig, dass sowohl vor Beginn des Arbeiten (bei Teams > 1) ein Pull durchgeführt wird. Ansonsten kann es passieren, dass auf einer veralteten Version gearbeitet wird, wodurch es zu Konflikten beim abschließenden Push kommen kann, welche dann händisch behoben werden müssen.
Der Pull 5) sorgt dafür, dass Änderungen seit dem letzten Pull lokal gemerged werden können. Sofern dies automatisch möglich ist, wird dies von Git getan, aber beispielsweise bei Word-Dokumenten ist dies nicht der Fall, da eine Änderung einer Zeile stets Auswirkungen auf das gesamte Dokument hat. (Siehe auch Aufgabe 7)

Alternative Quelle: https://git-scm.com/book/en/v1/Getting-Started

Gitignore:

Diese ist besonders wichtig, sollten Sie eine ganze Visual Studio Solution in das Repository einchecken. Hierfür einfach bei Google beispielsweise nach „visual studio gitignore“ suchen, und eine .gitignore mit passenden Ordner erstellen (beispielsweise Source). Ein Repository kann mehrere .gitignore-Dateien haben, und gelten immer für den Ordner in dem Sie sich befinden, und alle Unterordner. Dies ist wichtig, da durch das alleinige Öffnen von Visual Studio Änderungen in Dateien abgelegt werden. Da dies automatisch zu Konflikten führen würde, müssen solche Dateien ignoriert werden – was die Funktionsweise von VS keinerlei beeinflusst, da sie ja Lokal die Dateien besitzen. Ebenfalls sollten Build-Dateien und Produkte dieses Prozesses nicht übertragen werden, da dies ebenfalls Konfliktrisiko in sich birgt und diese zudem, mit vorschreitenden Projekt, auch relativ groß werden können.
Die Gitignore muss vor dem ersten pushen der jeweiligen Dateitypen angepasst sein. Wenn sie einmal beispielsweise die „hallo.tmp“ Datei gepushed haben, bleibt diese in der Versionskontrolle erfasst, auch wenn sie nachträglich die *.tmp Dateien ignorieren. Hierbei hilft nur ein löschen und pushen. Anschließend kann die Datei wieder hinzugefügt werden, und wird dann auch nicht weiter verfolgt.

Aufgabe 6 – Versionshistorie und Änderungen betrachten

Führen Sie hintereinander mehrere Änderungen an einer Datei durch. Der Ablauf im Projektalltag ist hierbei üblicherweise wie folgt:

- Beginn der Arbeit: **Pull**
- Nach logischen abgeschlossenen Einheiten wird ein **Commit** mit geeignetem Kommentar gemacht
    - Dies wird so lange wiederholt bis beispielsweise ein Ticket abgearbeitete ist
    - Prinzipiell sollten niemals viele offene Änderungen / Differenzen zwischen dem Lokalen und dem Remoterepository bestehen. Es gilt diese möglichst früh zu bearbeiten. Dies ist entweder durch **Commiten** und **Pushen**, zurücksetzen auf den Stand im Remoterepository oder durch Hinzufügen zur .gitignore möglich.
- Dann wird ein **Pull** durchgeführt, also alle Änderungen welche von Teammitgliedern getätigt wurden geholt.
    - Gegeben falls es gibt Konflikte, muss an dieser Stelle vereinigt (merged) werden
- Dann **Push**, also die lokalen Änderungen auf das Remote Repository übertragen.

Betrachten Sie innerhalb des „Log / Verlauf“-Reiters (ganz unten am Repository-Reiter in SourceTree) die Versionshistorie, Kommentare, Änderungen und lassen Sie sich die

[6] http://ndpsoftware.com/git-cheatsheet.html

Unterschiede für zwei Versionen dieser Datei darstellen - durch Markieren der unterschiedlichen Versionen, welche miteinander verglichen werden sollen.

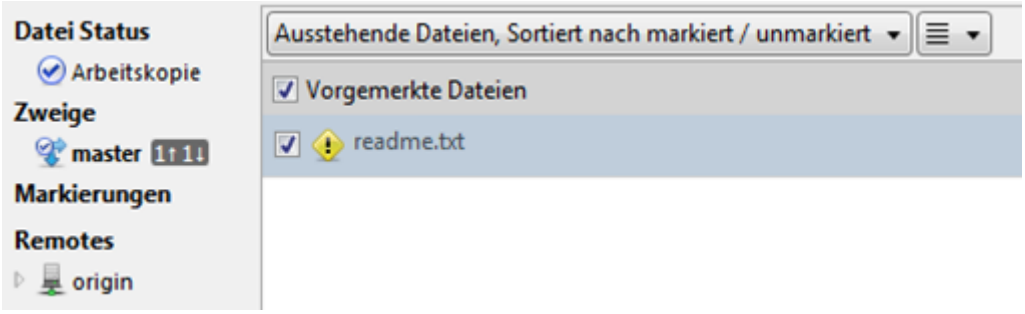
Benennen Sie die Datei um (z.B. zu „`readme.txt`“), checken Sie diese Datei ein und betrachten Sie danach die Versionshistorie der umbenannten Datei.

Aufgabe 7 – Konflikte und Vereinigen (merge)

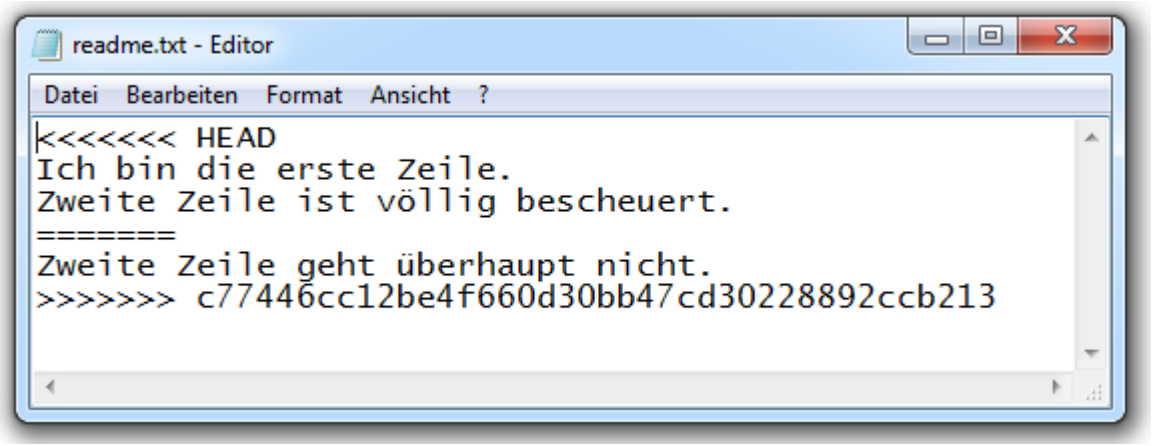
Zwei Teammitglieder modifizieren nun parallel die neue Datei (erst „**Pull**“ dann „**Push**“) und versuchen diese nacheinander zu übertragen. Was passiert beim zweiten Teammitglied? Untersuchen Sie folgende Fälle:

- Beide Teammitglieder modifizieren verschiedene Zeilen der Datei
- Beide Teammitglieder modifizieren dieselbe Zeile der Datei

Einer der beiden Fälle führt zu einem Konflikt, sollte nicht einfach behoben werden können:



Sowohl beim Öffnen der Datei „`readme.txt`“ als auch in SourceTree wird der Konflikt dargestellt, indem die Änderungen beider Teammitglieder im Text dargestellt werden. Die Textdatei sollte bei der geänderten Zeile etwa so aussehen:



Editieren sie nun die Datei geeignet, markieren Sie den Konflikt als gelöst (rechte Maustaste auf die Datei im SourceTree-Reiter Dateistatus) und dann „**Konflikt beheben**“. Hierfür kann auch ein externes Werkzeug zum Lösen von Konflikten verwendet werden, ein Tutorial für die Verwendung von externen Vereinigungstools finden sie hier[7].

Provozieren Sie erneut einen Konflikt, führen sie aber bei der Meldung des Konfliktes kein Update aus, sondern rufen sie stattdessen **„mit meinem Beheben**“ bzw. „**mit deren Beheben**“ auf. Was passiert, und wie wird ein Konflikt im Verlauf dargestellt?

[7] http://oliverbusse.notesx.net/hp.nsf/tutorial.xsp?documentId=C6A

Optional: Aufgabe 8 – Zusammenspiel von Gitlab und Git

Aufgabe 8.1 – Issue schließen mit Commit-Message

Versuchen Sie zum Abschluss, sich alle gegenseitig jeweils eine Aufgabe zuzuweisen, beispielsweise „Lege die Datei MaxReadme.txt“ an.“. Schließen Sie dieses Ticket, ohne die grafische Oberfläche von Gitlab zu verwenden, sondern mit Hilfe einer Commit-Message.

(Hinweis: siehe http://doc.gitlab.com/ee/)

Aufgabe 8.2 – Einbinden einer Datei aus Gitlab in das Wiki

Binden Sie die Datei „readme.txt“ immer mit dem aktuellen Stand, und eine ältere Version der Datei ein.

Tipp: Im Gitlab-Projekt ist es unter „Files“ möglich, durch das Repository zu Browsen.

Zusatzaufgaben:

1. Authentifizierung via SSH-Key

Für die Authentifizierung via SSH-Key. Gehen Sie wie folgt vor:

Sollten Sie noch keinen SSH-Key haben, werden Sie beim ersten Start von SourceTree gefragt, ob Sie einen Key generieren wollen. Alternativ lässt sich dies auch später erledigen, hierfür unter Tools auf SSH-Schlüssel importieren oder erstellen klicken:

- Auf Generate klicken.
- Maus im freien Feld unter dem Ladebalken bewegen (Zufallszahlenerzeugung)
    - Anschließend kann, wenn gewünscht, das Feld „**Key comment**“ geändert werden, wie „Max@Laptop“ oder was euch sinnvoll erscheint um den Key zuordnen zu können.
- Dann den privaten Schlüssel (an einer sicher Stelle) speichern – ob Passwortgeschützt ist hierbei optional.
- Den zugehörigen Public Key im oberen Teil des Fensters (beginnend mit „**ssh-rsa**“) vollständig kopieren und im Gitlab unter „**Profile Settings**“ und dann „**SSH Keys**“ hinzufügen.
- In SourceTree unter Tools -> Optionen bei SSH-Verbindungskonfiguration den Pfad zum privaten Schlüssel angeben.

Nach dem hinzufügen muss entweder SourceTree zusammen mit „Pageant“ - findet ihr in der Symbolleiste neben der Uhrzeit - beendet werden, oder alternativ Pageant öffnen und euren Key dort manuell hinzufügen. Bei jedem neuen Start von SourceTree wird auch Pageant gestartet und automatisch mit eurem Key versorgt.

Dann ebenso wie bei Aufgabe 4.1 in SourceTree auf „**Klonen / Neu**“ klicken, nur dieses Mal die „**SSH**“ URL verwenden:

git@inf-git.fh-rosenheim.de:sINFmamus/se1-inf-ue4-txx.git

2. Stash und Branch:

- Sehen Sie sich zusammen mit ihren Teammitgliedern das Cheatsheet aus Aufgabe 5 genauer an und überlegen Sie sich gemeinsam, wozu „Stash“ und das „local Repository“ gut sind.
- Wann macht es Sinn Zweige (also einen Branch) zu erstellen?