Übung 1: Projektdarstellung und Konfigurationsmanagement

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 ***Git*** und ***SVN*** (Subversion). Mit diesen Werkzeugen ist es möglich, Code (also ASCII-Texte) und binäre Dateien (z.B. 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 Projekt-Repositories

a) Bilden Sie Teams mit höchstens **sechs** Personen und bestimmen Sie je einen Teamadministrator für das Projekt-Repository.
b) Der Teamadministrator legt ein Projekt auf dem GitLab[1] an.
   - Als Login verwenden Sie ihre RZ-Kennung und Passwort (also wie beim Login auf der TH-Rosenheim Website)
   - Klicken Sie auf der Login-Seite auf den Button **New Project** und geben Sie für das Team den zugehörigen Projektnamen bei **Project path** ein.

     Projektname**:** **se-inf-20-txx** (txx = Team-Name Ihrer Wahl)

   - Geben Sie eine passende Beschreibung für das Projekt ein, und wählen Sie **Private** als **Visibility Level** aus. Das somit erstellte Projekt wird auf der Übersichtsseite angezeigt (je nachdem wie der globale Benachrichtigungslevel von GitLab eingestellt ist, erhalten Sie möglicherweise auch eine automatisch generierte E-Mail).

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 Settings→**Members** Teammitglieder hinzugefügt werden. Die Suche berücksichtigt sowohl die RZ-Kennungen (Anzeigename) als auch TH-Email-Adressen sowie Vor- und Nachnamen.

a) Weisen Sie den Teammitgliedern die notwendigen Nutzungsrechte zu (**Role Permission**).
b) Prüfen Sie (jedes Teammitglied!), ob Sie Zugang zu Ihrem Projekt haben. In Ihrem Email-Postfach sollte sich auch eine Benachrichtigung bezüglich Ihrer Projektzugehörigkeit befinden.

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

Aufgabe 3 – Erstellung Projekt-Homepage (Wiki)

Erstellen Sie gemeinsam in Ihrem Team eine Projekt-Homepage.

a) Halten Sie sich bei der Struktur an den typischen Inhalt eines Projekthandbuchs, z.B.

1 Projektstruktur
  1.1 Steckbrief
  1.2 Vorgehensmodell
  1.3 Teamstruktur
  1.4 Systemstruktur
2 Teamplan
3 Arbeitspakete und Aufgabenplan
  3.1 Aufgaben und Arbeitspakete
  3.2 Aufgabenplan
4 Beistellungen und Zulieferungen
5 Meilensteine und Auslieferungen
6 Risiken

Hinweise zur Syntax finden Sie hier[2].

b) Strukturieren Sie Ihr Wiki so, dass bei Kapitel ***1.3 Teamstruktur*** auf eine eigene Wiki-Seite verzweigt wird. Beschreiben Sie dort Ihr Team in Tabellenform mit den Spalten
- Name
- Rolle (Projektleiter, Entwickler, …) und
- Kontakt (= E-Mail Adresse).

Aufgabe 4 – Klonen des Repositorys

Jede Person Ihres Teams arbeitet mit einem vollständigen Projekt-Repository, hat also alle Versionen aller Dateien des Projekts in einem lokalen Verzeichnis. Über Git wird der Inhalt des lokalen Verzeichnisses mit dem zentralen (Remote) Repository abgeglichen. In dieser Übung lernen Sie jetzt die Grundbegriffe des Konfigurationsmanagements und wenden diese mit Git an.

a) Mit dem Anlegen des Projekts wurde automatisch ein Repository auf dem Git-Server erzeugt. Um nun Ihre Daten mit dem Repository zu synchronisieren, benötigen Sie ein passendes Tool.
Im Folgenden wird zwar die Benutzung mit ***TortoiseGit*** (Git GUI) beschrieben, die Verwendung anderer Tools wie z.B. *SourceTree* oder auch der Kommandozeile ist aber ohne weiteres möglich (eine Liste von Alternativen finden Sie hier[3], ein Tutorial für die Verwendung des Kommandozeilentools finden Sie hier[4]).

Auf den Labor-Rechnern steht *TortoiseGit* zur Verfügung. Installieren Sie es gegebenenfalls auf Ihrem Rechner oder ein anderes Git Tool Ihrer Wahl.

[2] https://inf-git.fh-rosenheim.de/help/user/markdown.md
[3] https://git-scm.com/downloads/guis
[4] https://try.github.io

b) Für jede Aktion mit dem Git-Server müssen Sie authentifiziert sein. Der schnellste Weg ist die Authentifizierung über Benutzername und Passwort. Hierfür wählen Sie in *TortoiseGit* **Git Clone…** (rechte Maustaste) aus, und fügen Sie die im Gitlab-Projekt angegebene „**https:**“ URL an, also beispielsweise:

https://inf-git.fh-rosenheim.de/sINFnnnnn/se-inf-20-txx.git

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

> ***Anmerkung**: Unterschied zwischen **Clone** und **Pull***
> - *Klonen wird zur Initialisierung Ihres Workspace / lokalen Repositorys verwendet.*
> - *Pull holt nur Änderungen aus dem Remote Repository.*

c) *(**Nicht im Rahmen dieser Übung**, aber …) Um das auf Dauer mühsame Eingeben von Benutzername und Passwort zu vermeiden, könnten Sie bei Interesse auch die Authentifizierung via SSH-Key bewerkstelligen.[5]*
Beim Git Clone-Dialog müssten Sie dann aber beispielsweise die „SSH“ URL `git@inf-git.fh-rosenheim.de:sINFnnnnn/se-inf-20-txx.git` *verwenden.*

Aufgabe 5 – Dateien dem Team zur Verfügung stellen

a) Ein Teammitglied erzeugt nun im Projektverzeichnis ein Unterverzeichnis `source` und darin eine Datei `.gitkeep`[6].

> ***Achtung**: 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.*

b) Zusätzlich soll im Wurzel-Verzeichnis noch eine Datei `HelloWorld.txt` angelegt werden. Diese Datei kann nach Erstellung den anderen Teammitgliedern zur Verfügung gestellt werden. Hierfür wählen Sie in *TortoiseGit* aus dem Kontextmenü **Git Commit** aus. Setzen Sie den Haken der Checkbox neben dem Verzeichnis und der Datei, schreiben einen Kommentar (Commit Message) und klicken auf **Commit**.

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

Mit **Commit** werden die Änderungen in das lokale Repository übertragen.

c) Um diese Änderungen nun auf das Remote Repository zu übertragen und damit den Teammitgliedern den Zugriff zu ermöglichen, muss ein **Push** durchgeführt werden (**Push…**). Bei dem Push-Dialog wählen Sie „**master**“ aus.

d) Alle Teammitglieder klonen sich nun das Projekt, falls noch nicht geschehen (Aufgabe 4b), beziehungsweise führen einen **Pull** durch (**Pull…**).
Danach sollte die erzeugte Datei `HelloWorld.txt` zusammen mit dem Ordner `source` und der `.gitkeep`-Datei für alle anderen Teammitglieder sichtbar sein.

[5] Anleitung z.B. unter https://doc.itc.rwth-aachen.de/display/SES/TortoiseGit+-+Git+auf+Windows
[6] http://stackoverflow.com/questions/7229885/what-are-the-differences-between-gitignore-and-gitkeep/7229996#7229996

Aufgabe 6 – 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[7]. Dies kann separat für jedes Verzeichnis angegeben werden.

a) Erstellen Sie im Ordner `source` eine `.gitignore` Datei mit dem Inhalt `*.tmp`. Fügen Sie jetzt zusätzlich eine Datei mit der Endung `.tmp` ein und beobachten Sie das Verhalten von SourceTree bei Ihrem nächsten Commit.

b) Probieren Sie nach Belieben! Vollziehen Sie den Ablauf

1. Hinzufügen von nicht vorgemerkten Dateien
2. Commit
3. Push

unter Zuhilfenahme des Cheatsheets[8] nach (alternative Quelle[9]).

Aufgabe 7 – Versionshistorie und Änderungen betrachten

a) Führen Sie hintereinander mehrere Änderungen an einer Datei durch. Der Ablauf im Projektalltag ist hierbei üblicherweise folgendermaßen:

1. Nach logisch abgeschlossenen Einheiten erfolgt ein **Commit** – mit geeignetem Kommentar!
   (Dies kann so lange wiederholt werden bis etwa ein „Ticket“ abgearbeitet ist.)
2. Dann wird ein **Pull** durchgeführt, um alle Änderungen zu holen, welche von Teammitgliedern getätigt wurden.
   (Dabei kann es zu Konflikten kommen und muss gegebenenfalls an dieser Stelle zusammengeführt werden – „**merge**“, siehe Aufgabe 8)
3. Schließlich erfolgt ein **Push**, um die lokalen Änderungen auf das Remote Repository zu übertragen.

b) Betrachten Sie die Versionshistorie, Kommentare und Änderungen (rechte Maustaste → TortoiseGit → **Show Log**) und lassen Sie sich die Unterschiede für zwei Versionen dieser Datei darstellen (Path-Fenster → rechte Maustaste auf Datei → **Compare with base**). Markieren Sie dazu unterschiedliche Versionen, welche miteinander verglichen werden sollen (**Compare revisions**).

c) Benennen Sie die Datei `HelloWorld.txt` um zu `readme.txt`, (TortoiseGit → **Rename…**) checken Sie diese Datei ein (**Commit**) und betrachten danach die Versionshistorie der umbenannten Datei.

[7] https://git-scm.com/docs/gitignore
[8] http://ndpsoftware.com/git-cheatsheet.html
[9] https://git-scm.com/book/en/v1/Getting-Started

Aufgabe 8 – Konflikte und Vereinigen (merge)

Zwei Teammitglieder modifizieren nun parallel die neue Datei (erst **Pull** dann **Push**) und versuchen diese nacheinander zu übertragen.

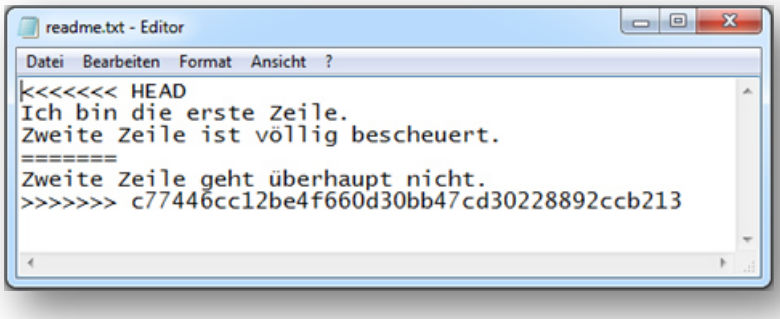
Was passiert beim zweiten Teammitglied?

a) Untersuchen Sie folgende Fälle:

1. Beide Teammitglieder modifizieren verschiedene Zeilen der Datei.
2. Beide Teammitglieder modifizieren dieselbe Zeile der Datei.

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

Sowohl beim Öffnen der Datei `readme.txt` als auch in *TortoiseGit* wird der Konflikt dargestellt, indem die Änderungen beider Teammitglieder im Text dargestellt werden. Die Textdatei könnte wegen derselben geänderten Zeile etwa so aussehen:



Editieren sie nun die Datei geeignet (rechte Maustaste → TortoiseGit → **Edit conflikts**), markieren Sie den Konflikt als gelöst (im TortoiseGitMerge Tool → „*Mark as resolved*“ oder außerhalb rechte Maustaste → TortoiseGit → **Resolve…**).

(Hierfür könnte auch ein externes Werkzeug zum Lösen von Konflikten verwendet werden. Ein Tutorial für die Verwendung von externen Vereinigungstools finden sie hier[10].)

(Commit anschließend nicht vergessen!)

b) Provozieren Sie erneut einen Konflikt!

Führen sie aber bei der Meldung des Konfliktes keinen Pull aus oder korrigieren manuell, sondern rufen sie stattdessen den Log auf (rechte Maustaste → TortoiseGit → **Show log**), selektieren die rot markierte Datei, und rufen **Resolve conflict using ‘mine‘** bzw. **Resolve conflict using ‘theirs‘** auf.

(Commit anschließend nicht vergessen!)

Was passiert, und wie wird ein Konflikt im Verlauf dargestellt?

[10] http://oliverbusse.notesx.net/hp.nsf/tutorial.xsp?documentId=9AA (für *SourceTree*)

Aufgabe 9 – Zusammenspiel von GitLab und Git

GitLab bietet natürlich eine Vielzahl an Möglichkeiten für Projektmanagement und Projektdarstellung. Wir wollen zum Abschluss nur noch ein paar wenige interessante Dinge ausprobieren …

a) Versuchen Sie zum Abschluss, sich gegenseitig eine Aufgabe (**Issue**) zuzuweisen, beispielsweise „Lege die Datei MaxReadme.txt an.“.

b) Mit GitLab/Git ist es praktischerweise möglich, Aufgaben gleich mit einem Commit als erledigt zu kennzeichnen
Schließen Sie eine solche zugewiesene Aufgabe, ohne die grafische Oberfläche von GitLab zu verwenden, sondern allein mit Hilfe einer Commit Message.

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

c) Oft ist es wünschenswert, auf relevante Dateiversionen direkt aus dem Projekt-Wiki heraus zuzugreifen. Auch dies ist mit GitLab möglich:
Versuchen Sie zum Abschluss, die Datei `readme.txt`
- mit dem aktuellen Stand
- eine ältere Version der Datei

in Ihr Projekt-Wiki einzubinden und nach Git zu verlinken.

*(Tipp: Im GitLab-Projekt ist es unter **Files** möglich, durch das Repository zu browsen.)*

d) Legen Sie schließlich im Projektverzeichnis Ihres Repository ein Unterverzeichnis `Übungen` an, und darin eine Datei `.gitkeep`.

(Dort können Sie künftig Ihre im Team erstellten Dokumente zu den Übungsaufgaben ablegen.)

Erstellen Sie noch in Ihrem Projekt-Wiki an geeigneter Stelle (siehe Aufgabe 3) einen geeigneten Link zu diesem `Übungen` Verzeichnis im Repository.