Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III

Übungen zur Vorlesung Softwaretechnologie

- Wintersemester 2019/20 -

Dr. Günter Kniesel

Übungsblatt 2

Zu bearbeiten bis: 18.10.2019, 16 Uhr

Bitte fangen Sie **frühzeitig** mit der Bearbeitung an, damit wir Ihnen bei Bedarf helfen können. Checken Sie die Lösungen zu den Aufgaben bitte in Ihr Repository ein, „Erklärungen“ bitte als Textdatei. Fragen zu Übungsaufgaben respektive zur Vorlesung können Sie auf der Mailingliste swt-tutoren@lists.iai.uni-bonn.de, bzw. swt-vorlesung@lists.iai.uni-bonn.de stellen.

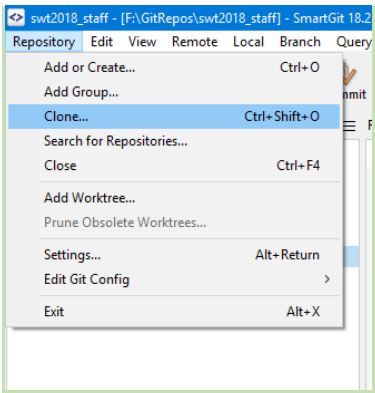
Praktische Aufgaben

Diese Aufgaben sollen von allen Gruppenmitgliedern einzeln bearbeitet werden, damit Sie sich genau mit dem Umgang mit Git vertraut machen können.

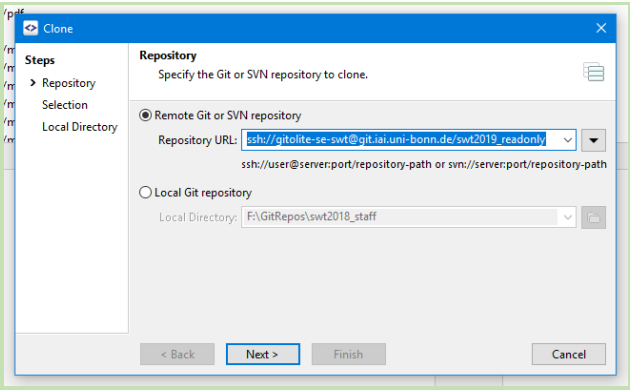
Aufgabe 1. *Branch, merge und vollständig dezentralisiertes Arbeiten* **11 Punkte**

Tipps zur Vorgehensweise in SmartGit sind mit SG gekennzeichnet.

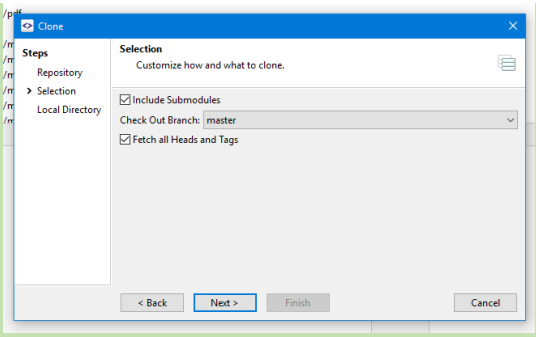
a) **(2 Punkte)** Clonen Sie ssh://gitolite-se-swt@git.iai.uni-bonn.de/swt2019_readonly. Danach sollten Sie in Ihrer lokalen Kopie die Datei "git.txt" vorfinden. Darin steht ein Text, der aus 40 Absätzen besteht. Suchen Sie nach dem Absatz, der Ihrer Gruppe zugeordnet ist. Jeder Absatz ist in drei Abschnitte unterteilt.



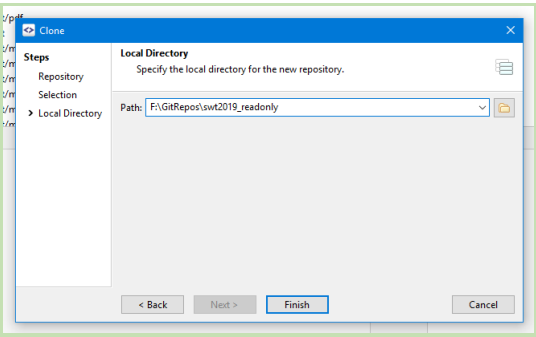
1. Repository clonen



2. SSH Link zum Gruppen



3. master-Branch auswählen



4. Speicherort festlegen

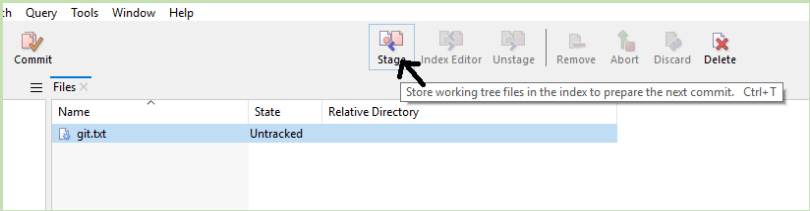
Jeder von Ihnen soll *parallel zu den anderen Gruppenmitgliedern* auf seinem eigenen Rechner genau einen Abschnitt dieses Absatzes bearbeiten. Ordnen Sie sich selbst die Abschnitte zu, indem Sie Ihre Vornamen alphabetisch sortieren. Der 1. Vorname bearbeitet den 1. Abschnitt, der 2. Vorname den zweiten Abschnitt, und so weiter. Wenn Sie mehr als 3 Teammitglieder sind, geht es zyklisch weiter. Dann gibt es mehrere Leute, die den gleichen Abschnitt bearbeiten.

b) **(3 Punkte)** Führen Sie folgende Änderungen durch:

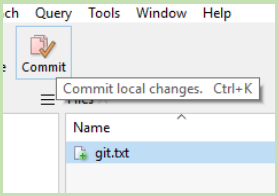
1) Entsprechend Ihrer vereinbarten Aufgabenteilung pro Abschnitt, übersetzen Sie "Ihren" Abschnitt des Absatzes ihrer Gruppe ins Deutsche. Es kommt hierbei nicht auf grammatikalische und orthographische Korrektheit an; dennoch sollte Ihre Übersetzung möglichst sinnig sein.

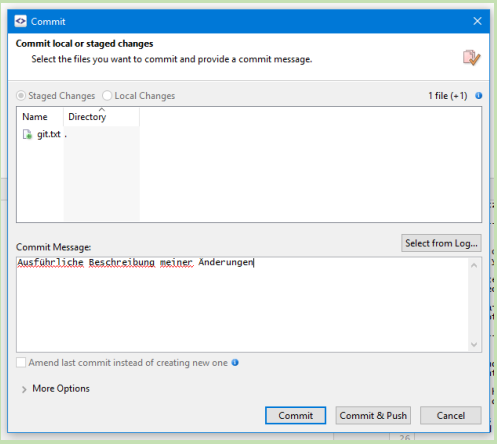
Wie man Englisch übersetzt und eine Datei bearbeitet, sollten Sie bereits wissen ;)

2) Committen Sie die Änderungen in Ihr lokales Repository.



1. Datei anwählen und stagen





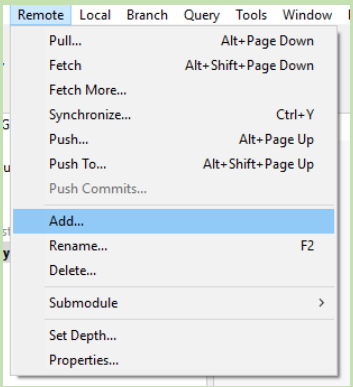
2. Commit anklicken 3. Beschreibung hinzufügen und Commit drücken

Sie können nun Ihre Änderungen nicht in das „ReadOnly“ Repository zurück pushen, da Sie dazu sinnvollerweise keine Rechte haben (sonst würden diejenigen, die das Repository später klonen, nichts mehr zum übersetzen haben). Stattdessen pushen Sie Ihre Änderungen in das Repository ssh://gitolite-se-swt@git.iai.uni-bonn.de/swt2019_read_and_write.

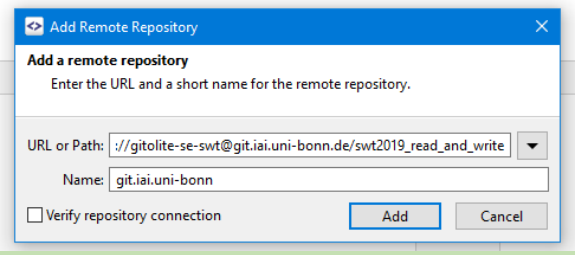
Dazu müssen Sie als erstes:

c) **(2 Punkte)** Remote und tracking neu setzen:

1) Das ReadWrite-Repository als neuen Remote eintragen.
SG: „Remote“ > „Add…“ > *URL eintragen* > *Name für den Remote eintragen, z.B. ReadWrite* > „Verify repository connection“ *weglicken* > „Add“.



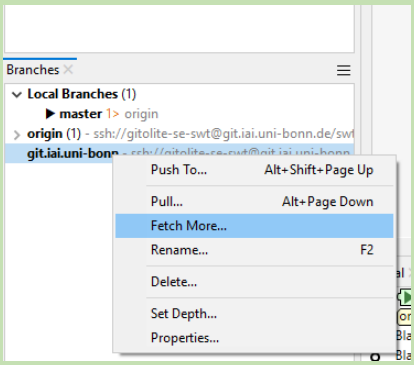
1. Remote add…



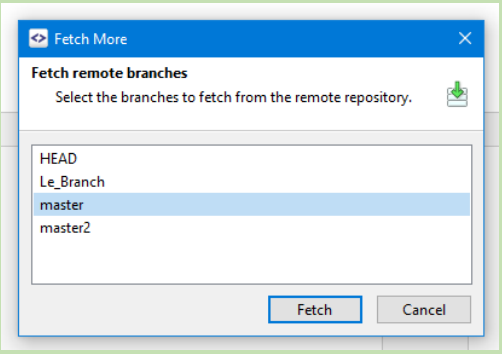
2. URL eintragen und Häkchen entfernen, Add

2) Anschließend Git mitteilen, dass Ihr Master-Branch nun den master-Branch im neuen Remote, dem ReadWrite-Repository, „tracken“ soll.

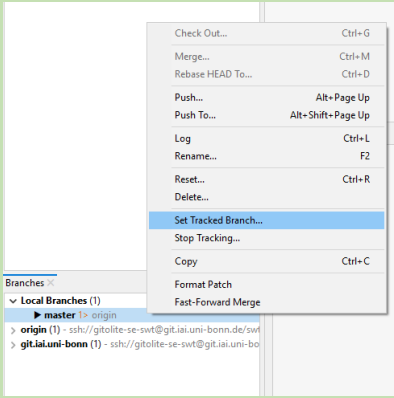
SG: Im „Branches“-View Rechtsklick auf den Remote → „Fetch More…“ → Rechtsklick auf *lokalen* master-Branch → „Set Tracked Branch“ → Master aus ReadWrite-Repository auswählen).

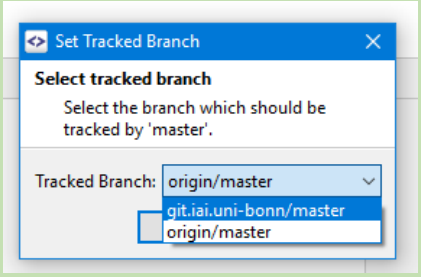


1. Im Branches View fetch more…



2. master des Remotes wählen

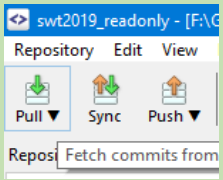




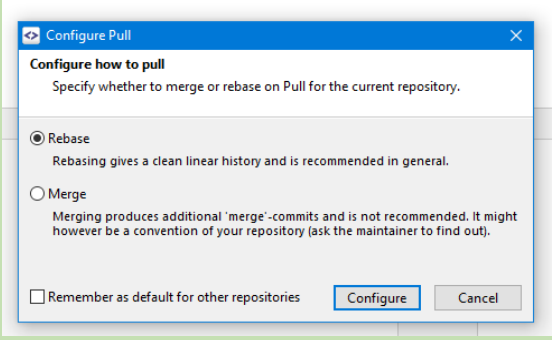
3. Set tracked Branch auf lokalem Master auswählen 4. Master des Remotes

d) **(4 Punkte)** Versuchen Sie nun zu pushen (SG: "Push"). Falls Sie der/die Schnellste waren, geht das problemlos. Ansonsten wird Git eine Fehlermeldung ausgeben. Dann müssen Sie stattdessen:

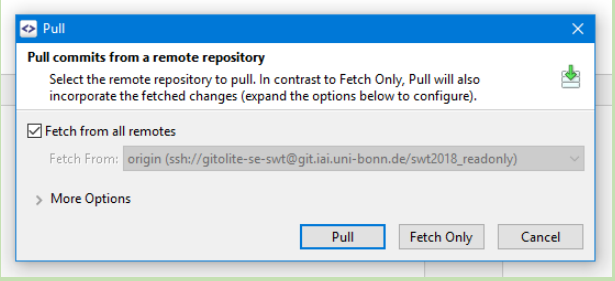
1) Die Änderungen, die Ihre Kollegen schon gepusht haben, zu sich pullen (SG: "Pull"). Git führt dabei automatisch einen "Merge" durch (SG: Sie können in den Optionen des "Pull"-Dialogfensters steuern ob ein "Merge" oder ein "Rebase" durchgeführt wird).



1. Pull drücken



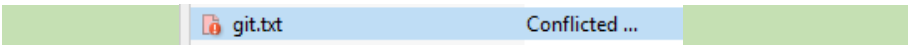
2.Rebase oder Merge konfigurieren



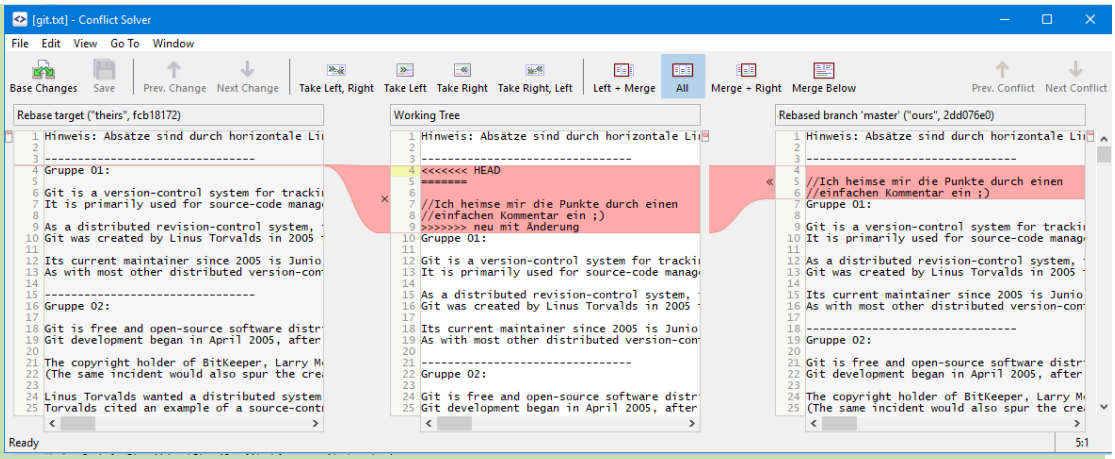
3. Pullen

2) Lösen Sie eventuell auftretende Konflikte. Achten Sie genau darauf, wann und warum Konflikte auftreten. Diese Beobachtungen brauchen Sie später zur Lösung von Aufgabe 3. Sprechen Sie gegebenenfalls mit Ihrem am Konflikt beteiligten Kollegen, wenn unklar ist, welche Konfliktlösung Sinn ergibt.

- SG: Konflikte werden durch ein Dateisymbol mit einem roten Punkt, in dem sich ein Ausrufezeichen befindet, angezeigt.



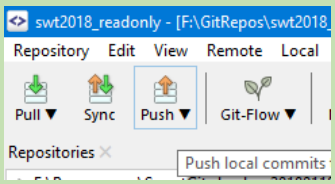
- SG: Den Konfliktlösungseditor von SmartGit starten Sie durch Doppelklick auf die betroffene Datei oder durch Rechtsklick → Conflict Solver. Darin können Sie alle Änderungen der Datei nachverfolgen und die Konflikte auflösen indem Sie pro Änderung entscheiden, welcher Stand (ihrer oder der des Anderen) übernommen wird.



Im Konfliktlöser für jede widersprüchliche Stelle eine Lösung finden

3) Wenn alle Konflikte in der Datei gelöst sind, speichern Sie die Datei und führen ein „Stage“ durch. Dadurch wird Git mitgeteilt, dass der neue Stand als Lösung des Konflikts betrachtet werden und Teil des nächsten „Commit“ sein soll. (SG: Der Konfliktlösungseditor fragt beim Speichern immer, ob für die Konfliktlösung ein "Stage" durchgeführt werden soll. Antworten Sie mit "Ja".)

4) Anschließend committen Sie. Dadurch werden die durch das „stage“ kumulierten Konfliktlösungsänderungen (es kann ja sein, dass Sie viele Konflikte in vielen Dateien haben) gemeinsam ins Repository übertragen. Versuchen Sie nun erneut zu pushen. Sofern nicht erneut jemand schneller war sollte es nun funktionieren. Ansonsten müssen Sie noch mal 1) – 4) durchführen.



Pushen drücken

Aufgabe 2. *Dateioperationen mit Git* **13 Punkte**

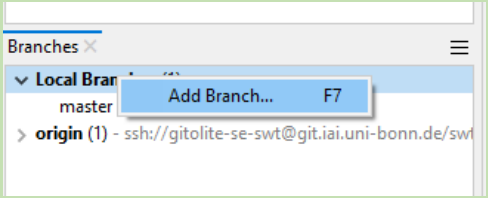
In dieser Aufgabe erstellen Sie einen eigenen Branch in Ihrem Gruppen-Repository und erzeugen darin Dateien, benennen sie um und löschen manche wieder. Um Ihr Gruppenrepository zu clonen, nutzen Sie die folgende URL und ersetzen Sie „NN“ durch Ihre Gruppennummer:

ssh://gitolite-se-swt@git.iai.uni-bonn.de/swt2019_gruppe_NN
Achten Sie bitte darauf, nun Ihren neu generierten **Private-Key** zu nutzen!

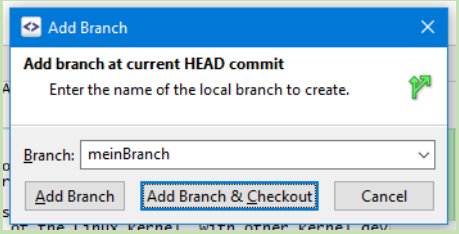
a) **(1 Punkt)** Erzeugen Sie in dem Ordner Ihres Gruppen-Repositories einen Ordner „Dateien_*Vorname*“, darin die Dateien „Datei 0.txt“ und „Datei 1.txt“ mit beliebigem, nicht-leeren Inhalt.

b) **(1 Punkt)** Führen Sie einen Commit der beiden Dateien durch.

c) **(1 Punkt)** Erzeugen Sie lokal einen Branch und wechseln Sie in diesen Branch. (SG: *Branch → Add Branch… → Add Branch & Checkout*).

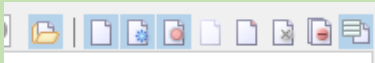


1. Branch hinzufügen



2. Branch Checkout

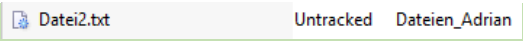
d) **(1 Punkt)** Fügen Sie (auf Systemebene) in dem Ordner eine neue Datei „Datei 2.txt“ hinzu. Wechseln Sie nach SmartGit. Sehen Sie die neue Datei? Sehen Sie die alte Datei auch noch? Finden Sie heraus worüber Sie steuern können, ob sie alle Dateien / neue Dateien / veränderte Dateien im SmartGit-Fenster "Files" sehen.



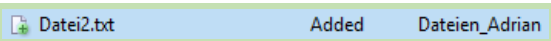
Von links:

- Dateien aus Unterordnern
- Alle Dateien (auch unveränderte)
- Alle Dateien, die noch nicht unter Versionskontrolle stehen
- Alle Dateien, die gestaged wurden
- Die weiteren Icons sind für unsere Zwecke nicht von großer Bedeutung

e) **(1 Punkt)** Welches Symbol hat die neue Datei? Teilen Sie Git mit, dass sie auch ins Repository übernommen werden soll, indem Sie die Datei auswählen und einen "Stage" durchführen. Wie verändert sich das Dateisymbol?

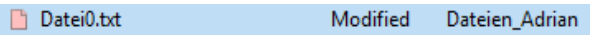


Datei nach Erstellung



Datei nach Staging

f) **(1 Punkt)** Schreiben Sie die Antworten zu d) und e) in die erste Datei (Datei 0.txt). Wie verändert sich das Dateisymbol nach dem Speichern? Schreiben sie auch die Antwort hierzu in die Datei.



g) **(1 Punkt)** Führen Sie nun einen Commit, mit den Änderungen, durch.

h) **(1 Punkt)** Benennen Sie nun (auf Systemebene) „Datei 0.txt“ in „Antworten.txt“ um, "Datei 1.txt" in "Eins.txt" und "Datei 2.txt" in "Zwei.txt". Was bemerken Sie, wenn Sie nach SmartGit zurückkehren? (Alternativ: Was bemerken Sie, wenn Sie auf der Kommandozeile einen „git status“ Befehl ausführen?) Wie lässt sich Ihre Beobachtung erklären? Schreiben Sie es in die Datei „Antworten.txt“ und führen Sie einen weiteren Commit durch, der alle Änderungen beinhaltet (SG: Achten Sie darauf beim Commit alle Dateien, die sich geändert haben zu selektieren.).

Antworten.txt	Renamed (untracked)	Dateien_Adrian
Eins.txt	Renamed (untracked)	Dateien_Adrian
Zwei.txt	Renamed (untracked)	Dateien_Adrian

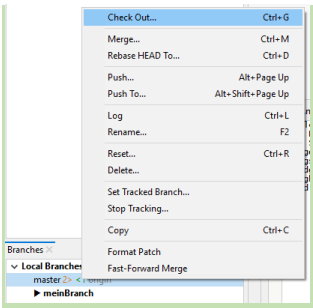
Die Umbenennungen wurden erkannt

i) **(1 Punkt)** Führen Sie folgende Schritte in einem Commit durch: Ändern Sie den Dateinamen von „Antworten.txt“ in „Antworten_final.txt“. Anschließend schreiben Sie folgende Überschrift in Ihre Datei: „Finale Antworten [DATUM VON HEUTE]“. Was fällt Ihnen auf? Schreiben Sie Ihre Antworten ebenfalls in „Antworten_final.txt“ und committen Sie letztendlich alle Ihre Änderungen.

Antworten.txt	Missing	Dateien_Adrian
Antworten_final.txt	Untracked	Dateien_Adrian

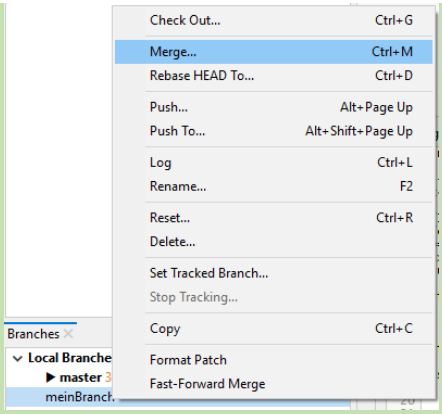
Die Datei wird nicht als umbenannt und verändert, sondern als gelöscht und neu erstellt angesehen. Dies passiert aber erst ab einem gewissen Prozentsatz an Änderungen! Es kann also auch passieren, dass Ihre Datei als umbenannt angesehen wird.

j) **(1 Punkt)** Wechseln Sie in den Haupt-Branch (master) zurück, wählen Sie dabei beim Checkout, *falls* Sie danach gefragt werden „save stash“ aus. Ändern Sie den Inhalt von "Datei 1.txt".



In master Branch wechseln

k) **(1 Punkt)** Führen Sie ein "merge" mit dem Stand des anderen Branches durch. Vergleichen Sie genau was problemlos geht und wo Konflikte auftreten. Achten Sie vor allem darauf, wie Git mit den beiden umbenannten Dateien umgeht. Können Sie den Unterschied erklären?



master mit anderem Branch mergen. Rechtsklick auf den anderen Branch

Die Dateien „Datei0.txt“ und „Datei1.txt“ sind weg, dafür sind die anderen Dateien da. Dies liegt an dem Commit mit der Umbenennung aus dem anderen Branch.

l) **(1 Punkt)** Committen Sie und pushen Sie das Ergebnis in ihr Gruppen-Repository.

m) **(1 Punkt)** Laden Sie nun noch die Notizen zu Blatt 1, Aufgabe 2 in Ihr Gruppen-Repository.

Theoretische Aufgaben

Aufgabe 3. *Git Konflikte* **5 Punkte**

Schauen Sie sich im Konfliktlösungseditor von SmartGit oder den Meldungen auf der Git-Bash-Kommandozeile genau um, so dass Sie verstehen, wann genau ein Konflikt auftritt. Versuchen Sie insbesondere zu verstehen welche der folgenden Fälle in Git zutreffen oder nicht:

1) Wenn die gleiche Datei von unterschiedlichen Leuten bearbeitet wurde, gibt es einen Konflikt.
2) Wenn die gleiche Zeile einer Datei von unterschiedlichen Leuten bearbeitet wurde, gibt es einen Konflikt.
3) Wenn in der gleichen Zeile von unterschiedlichen Leuten unterschiedliche Änderungen durchgeführt wurden, gibt es einen Konflikt.
4) Wenn … *<Ihre eigene Formulierung, falls keine der obigen passt>* …

Definition 3 ist korrekt.

Erzeugen Sie eine Datei "Konfliktdefinition.txt". Schreiben Sie darin Ihr Fazit auf. Wenn Sie der/die Erste waren (also Schritt (d1) bis (d5) aus Aufgabe 1 nicht durchführen mussten), schreiben Sie auf, was Sie aus der Vorlesung hinsichtlich des Konfliktbegriffes verstanden haben.

Committen Sie die Datei in IHR GRUPPENREPOSITORY und pushen Sie sie. Falls andere schneller waren, gehen Sie wie in Punkt 1d vor. Falls Konflikte auftreten, überschreiben Sie nicht die Vorschläge der Anderen, sondern sorgen Sie bei der Konfliktlösung dafür, dass Ihr

Formulierungsvorschlag als erkennbarer eigener Abschnitt unter den anderen steht (fügen Sie z.B. den eigenen Namen vor dem Abschnitt hinzu).

Aufgabe 4. *Git Vokabelliste* **8 Punkte**

Erklären Sie kurz die folgenden Begriffe:

a) Push und Pull

Neue Commits in eines der Remote Repositories laden bzw. aus einem der Remote Repositories holen.

b) Commit

Änderungen ins lokale Repository übernehmen.

c) Branch

Entwicklungszweig. Ermöglicht parallele verschiedene Entwicklungslinien, welche man hinterher zusammenführen kann. Technisch (in git) eine Folge von commits, wo jeder neuere auf seinen Vorgänger zeigt und ein Zeiger auf das neueste Element der Folge.

d) Merge und Rebase

Merge: Zusammenführen von verschiedenen Branches / Entwicklungslinien.
Rebase: Chery Picking aller Commits aus anderem Branch in den aktuellen Branch, in der Reihenfolge, wie sie in dem anderen branch angeordnet waren.

e) Working Directory

Der Ordner in dem der .git-Ordner enthalten ist. Er enthält ferner die Dateien die aus dem Repository zurzeit „ausgechecked“ sind und evtl. Dateien, die nicht unter versionskontrolle stehen.

f) Stage/Index

Veränderte Dateien, die in den nächsten Commit übernommen werden sollen.

g) Remote Repository

Anderes Repository, in welches man Änderungen pushen oder von welchem man Änderungen pullen kann. Falls das aktuelle Repository von einem anderen gecloned wurde, ist das Ursprungsrepository einer der Remotes und heisst laut Konvention „origin“. Wichtig hierbei: Es muss nicht zwangsläufig ein Remote Repository geben. Es kann keines geben (wenn wir nur lokal arbeiten) und es kann auch mehrere geben.

h) Tracking und Fetch (mit Zusammenhang)

Fetch lädt den Zustand eines Remote Repository incl. Aller vorhandenen Branches und commits als Kopie ins lokale Repository.
Durch das Setzen vom **Tracking** eines lokalen Branches teilt man Git mit, welchem Branch dieser im Remote entsprechen soll.

Aufgabe 5. *Arbeiten mit Repositories* **6 Punkte**

a) **(1 Punkt)** Erklären Sie die wesentlichen Unterschiede zwischen zentraler und verteilter Versionskontrolle.

Zentrale Versionsverwaltung:
- Ein Repository
- Interaktion über zentralen Server

Verteilte Versionsverwaltung:
- Gleichberechtigte Repositories
- Zentrale Instanz organisatorisch möglich

b) **(1 Punkt)** Erklären Sie den Unterschied zwischen erfolgreichem Merge und Merge-Konflikt.

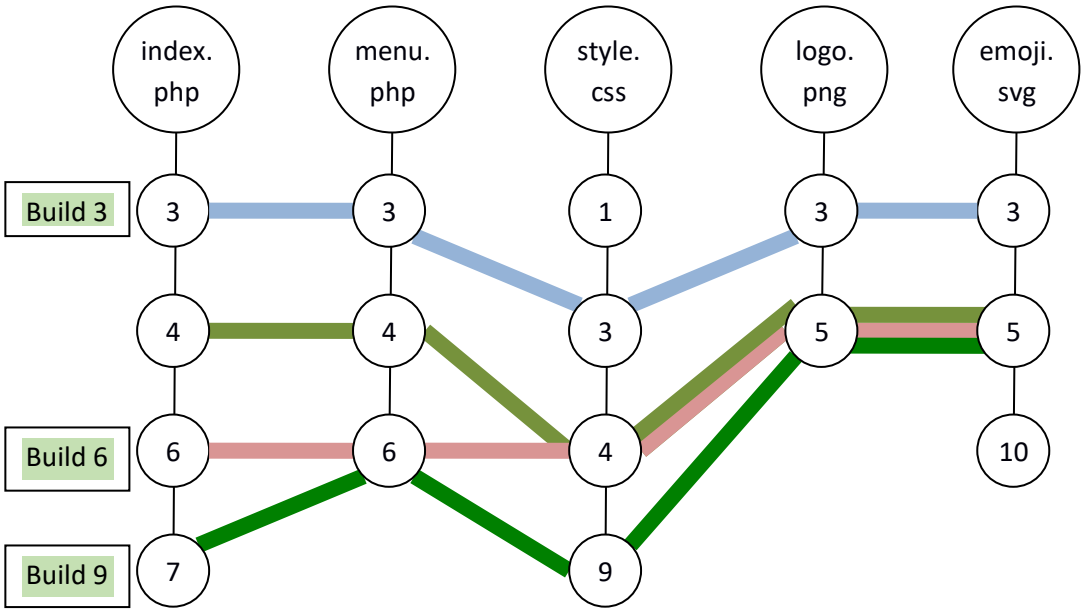
Erfolgreicher Merge: Die beiden branches können ohne Konflikt zusammengeführt werden.
Merge Konflikt: Es wurden in den beiden zu integrierenden Branches an der gleichen Stelle (Datei, Zeilein Datei) widersprüchliche Änderungen gemacht, sodass manuell die „korrekte“ Variante gewählt werden muss.

c) **(2 Punkte)** In Folie 2-28 sind Änderungen genannt, für die SCM-Systeme nicht erkennen können, dass sie für die Programmsemantik irrelevant sind. Benennen Sie jede Kategorie, erläutern Sie sie und geben Sie jeweils ein eigenes Beispiel von korrektem Java Code an, das den Grenzfall verdeutlicht.

Siehe besonders Folie 2-28 „Grenzen der Änderungserkennung und darauf basierender Konfliktauflösung“. Möglicherweise irrelevante Änderungen betreffen:
- *Kommentare*: Das Hinzufügen oder Entfernen von Kommentaren
- *Formatierungsänderungen*: Hinzufügen von Tabs, Klammern in nächster Zeile, Leerzeichen statt Tabs,…
- *Bruchstücke von Namen*: Variablennamen nur zum Teil ändern, besonders auch ändern von Variablennamen in Variablen mit gleichen Buchstaben an gleicher Stelle (**B**au**m** > **M**au**s**, führt zur Markierung jeden einzelnen neuen Buchstabens)
- *Verschiebungen*: Konstruktor vom Anfang ans Ende der Klasse schieben
- *Haupt-/Folgeänderungen*: m() in doM() umbenennen. Jeder Aufruf von m() wird infolgedessen ebenfalls in doM() umbenannt. Solche Folgeänderungen werden als eigene Änderung angezeigt, statt nur die Hauptänderung (Umbenennung von m()) anzuzeigen.

d) **(2 Punkte)** Gegeben sei folgende Abbildung mit verschiedenen Dateiversionen. Markieren Sie, welche Commits zu den Build-Nummern 3, 6 und 9 gehören. Welche Nummer hat der Build, der bereits in der Abbildung eingezeichnet ist?

Der vorgezeichnete Build hat die Nummer 5.


index.
php
menu.
php
style.
css
logo.
png
emoji.
svg
Build 3
Build 6
Build 9
3
3
1
3
3
4
4
3
5
5
6
6
4
10
7
9

Aufgabe 6. *Software Configuration Management FAQ* **4 Punkte**

Beantworten Sie die folgenden Fragen (ggf. mit kurzer Begründung).

a) **(1 Punkt)** Was ist ein atomarer Commit?

Ein atomarer Commit wird komplett oder gar nicht durchgeführt, damit z.B. bei Unterbrechung des Commits durch Netzwerkprobleme das Repository nicht in einem inkonsistenten Zustand hinterlassen wird. Datenbankbasierte SCM wie z.B. SVN setzen Commits als Transaktionen um.

b) **(1 Punkt)** Was sind Check-outs und Check-ins?

In SVN und anderen zentralen Systemen: Erstellung einer Arbeitskopie vom Repository (check-out) bzw. ein Projekt dem Repository hinzufügen (check-in).

c) **(1 Punkt)** Wird das Umbenennen eines Verzeichnisses auf Systemebene von Git als Umbenennung erkannt?

Git erkennt eine Umbenennung auf System-Ebene, wenn der Inhalt gar nicht (gleicher Hash) oder nur bis zu einem bestimmten Prozentsatz verändert wurde.

Sind sowohl Name, als auch größere Anteile der jeweiligen Datei geändert, erkennt Git dies als löschen der Datei mit altem Namen und erstellen der Datei mit neuem Namen.

d) **(1 Punkt)** Ein Commit lädt einen Datensatz ins Remote-Repository. Richtig?

Nein, ein Commit speichert den Datensatz im lokalen Repository ab.

∑ 47 Punkte