Übungsblatt 1 – Ausgabe am 27.04.2020

Abgabe der Lösungen bis 04.05.2020, 12:00 Uhr, per Mail.

Aufgabe 1.1: Versionsverwaltung

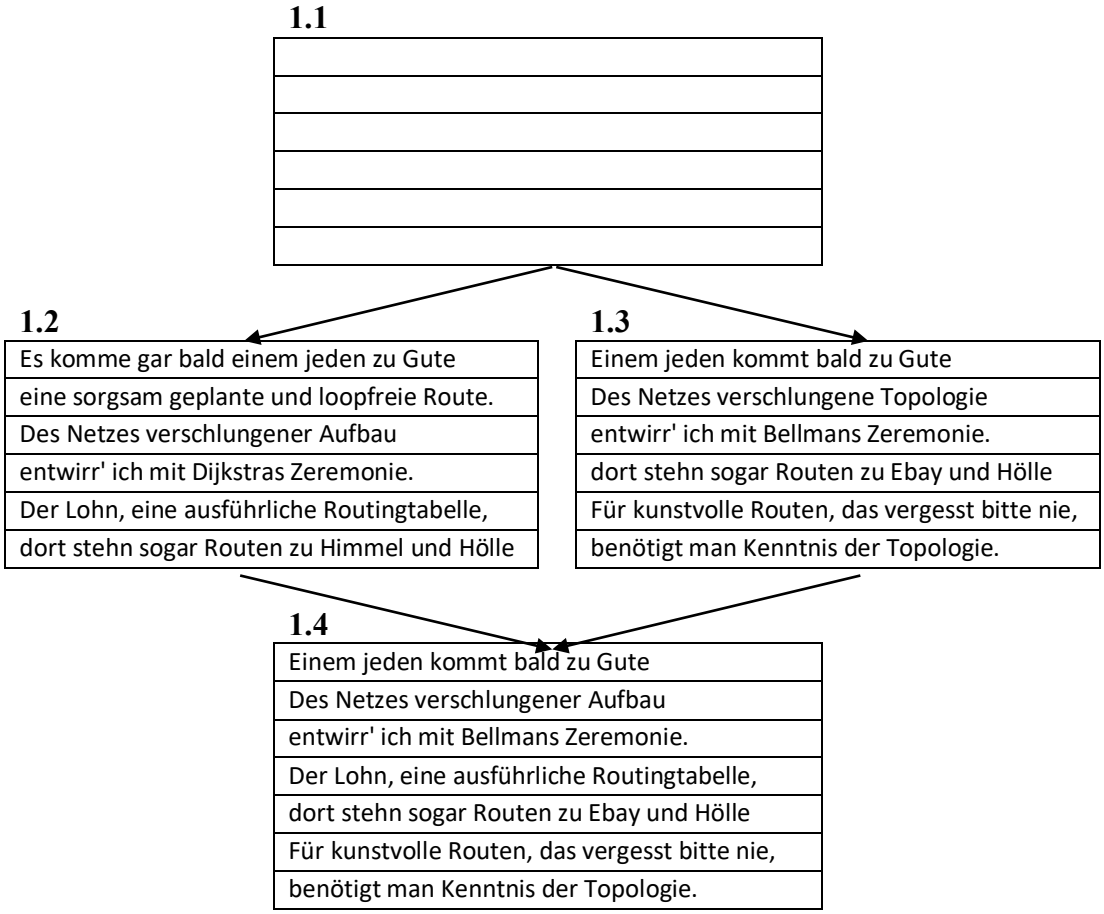
a) Erstellen Sie manuell eine diff-Datei (Änderungen von Datei1.txt zu Datei2.txt) gemäß der Notation aus der Übung für die beiden untenstehenden Texte. Benutzen Sie eine Kontextzeile vor den Änderungen. Setzen Sie dabei die vorgestellten Operationen (einfügen, löschen, ersetzen) sinnvoll ein.

	Datei1.txt		***Datei2.txt***
01	Wer routet so spät durch Nacht und Wind?	01	Wer routet so spät durch Nacht und Wind?
02	Es ist der Router, er routet geschwind!	02	Es ist die FritzBox, sie routet geschwind!
03	Bald routet er hier, bald routet er dort	03	Jedoch die Pakete, sie kommen nicht fort.
04	Jedoch die Pakete, sie kommen nicht fort.	04	Sie sammeln und drängeln sich, warten recht lange
05	Sie warten und warten	05	in einer zu niedrig priorisierten Schlange.
06	da meldet sich vorlaut der Routingprozeß	06	Die Schlangen sind voll, der Router im Streß,
07	und ruft: "All Ihr Päckchen, Ihr sorgt Euch zu viel,	07	da meldet sich vorlaut der Routingprozeß
08	nicht der IP-Host, nein, der Weg ist das Ziel!"	08	und ruft: "All Ihr Päckchen, Ihr sorgt Euch sehr viel“

b) Wie sieht Datei3.txt aus, wenn sie folgenden Patch auf Datei2.txt anwenden?

```
*** 01,02 *** Hunk 1 ****
      Wer routet so spät durch Nacht und Wind?
-     Es ist die FritzBox, sie routet geschwind!
--- 01,03 ---
      Wer routet so spät durch Nacht und Wind?
+     Es ist der Router, er routet geschwind!
+     Bald routet er hier, bald routet er dort
**********************
*** 07,08 *** Hunk 2 ****
      da meldet sich vorlaut der Routingprozeß
!     und ruft: "All Ihr Päckchen, Ihr sorgt Euch sehr viel“
--- 08,10 ---
      da meldet sich vorlaut der Routingprozeß
!     und ruft: "All Ihr Päckchen, Ihr sorgt Euch zu viel,
+     nicht der IP-Host, nein, der Weg ist das Ziel!"
**********************
```

c) Betrachten Sie die verschiedenen Versionen der untenstehenden Texte. Version 1.2 wurde zuerst erstellt, Version 1.3 folgte. Beide basieren auf Version 1.1. Version 1.4 ist **ausschließlich** durch *Drei-Wege-Verschmelzung* aus den Versionen 1.1, 1.2 und 1.3 entstanden. Rekonstruieren Sie die verloren gegangene Version 1.1, welche sich möglichst wenig von Version 1.4 unterscheidet.



Aufgabe 1.2: Teamarbeit mit git

- Diese Aufgabe ist für Gruppen mit maximal vier Personen geeignet. Sie können die Aufgabe aber auch in kleineren Gruppen oder allein bearbeiten, in dem Sie mehrere Rollen zugleich einnehmen.
- Ein Mitglied der Gruppe ist Projektleiter **P**. Die restlichen Gruppenmitglieder sind **A**, **B** und **C**.
- Es wird empfohlen, die Aufgabe mit msysgit (https://msysgit.github.com/) und TortoiseGit zu lösen (http://code.google.com/p/tortoisegit/).
- Geben Sie beim Commit immer im Kommentar an, was Sie genau geändert haben.
- Zur Bearbeitung dieser Aufgabe erstellen Sie bitte die Datei *gedicht.txt* mit dem unten aufgeführten Inhalt. Diese Datei enthält vier Fehler (umgekehrte Wörter), diese werden nach und nach behoben (simuliert das Beheben von Bugs in einem Programm).
- Halten Sie die durch die Schritte vorgegebene zeitliche Reihenfolge ein.
- Es empfiehlt sich bei der Bearbeitung der Aufgaben den Teammitgliedern „über die Schulter zu schauen“ (z. B. via Screen-Sharing), um sämtliche Schritte einmal gesehen zu haben.

- Die zu erstellenden Logmessages und Screenshots dienen dann als Ihre Abgabe für diese Aufgabe (bitte in einer zip-Datei abgeben).

1. Alle Teammitglieder legen sich einen Account bei GitHub an (https://github.com/).

2. **P** legt ein neues Repository auf GitHub an (es sind nur die Schritte unter „Make a new repository on GitHub“ auszuführen: https://help.github.com/articles/create-a-repo). Weiterhin fügt **P** **A**, **B** und **C** als Collaborators hinzu:
   - Repository auswählen
   - Settings anklicken (Symbol mit Schraubenschlüssel und Schraubendreher)
   - Collaborators anklicken
   - **A**, **B** und **C** über ihre GitHub-Benutzernamen hinzufügen

3. **P** erzeugt eine lokale Kopie des Repositories durch Klonen (Kontextmenü „Git clone…“, URL: https://github.com/<benutzername>/<repositoryname>.git). Nun legen Sie einen Order trunk im Arbeitsverzeichnis an und kopieren Sie die Datei *gedicht.txt* in diesen Ordner. Dann committen Sie (Kontextmenü „Git Commit -> „master“…“). Master ist in diesem Fall der Master-Branch, also der Hauptentwicklungszweig.
   Sollte nach *user name* und *email* gefragt werden, klicken Sie auf *yes*. Wählen sie *local* aus und entfernen sie die Häkchen bei *Inherit* neben *Name* und *Email*. Tragen Sie nun Ihren GitHub-Benutzernamen und Ihre Email-Adresse in die entsprechenden Felder ein.
   Im Commit-Dialog muss die Datei *trunk/gedicht.txt* ausgewählt werden. Überprüfen Sie das Ergebnis, indem Sie den Repository-Browser betrachten (Kontextmenü „TortoiseGit“ → „Repo-browser“).

4. **A**, **B** und **C** erzeugen sich jetzt ebenfalls eine Kopie des Repository. Sie stellen fest, dass die Datei *gedicht.txt* gar nicht in ihrer Kopie vorhanden ist, obwohl **P** die Datei bereits committet hat. Dies liegt daran, dass **P** die Datei nur in seiner lokalen Kopie des Repository’s committet hat.

5. Um die Datei *gedicht.txt* an alle Teammitglieder zu verteilen, muss **P** die Datei aus seiner lokalen Kopie in die Kopie, welche auf dem GitHub-Server liegt, pushen (Kontextmenü „TortoiseGit → Push…“).

6. **A**, **B** und **C** holen sich nun die Datei *gedicht.txt*, indem sie die letzten Änderungen am GitHub-Repository pullen („TortoiseGit → Pull…“). Nun sollten alle einen Ordner *trunk* in ihrer lokalen Kopie sehen können, der die Datei *gedicht.txt* beinhaltet.

7. **A** öffnet die Datei *gedicht.txt* und behebt den ersten Fehler (Zeile 4 „tethcuelre“) und committet und pusht die Lösung (nachdem **B** und **C** gepullt haben und bevor **B** und **C** ihre Lösungen committet und gepusht haben). Dokumentieren Sie das Ergebnis durch einen Screenshot der Logmessages (Kontextmenü „TortoiseGit“ → „Show log“).

8. **B** pullt sich die Änderungen von **A**. Nun öffnet **B** die Datei *gedicht.txt* und behebt den zweiten Fehler (Zeile 14 „suanih“) und committet die Lösung und pusht sie (nach Person **A** jedoch vor Person **C**). Um die Unterschiede zur vorherigen Version herauszufinden, vergleicht **B** die letzten beiden Versionen (die Änderung von **A** und **B**) miteinander, indem Sie sich die Logs anzeigen lassen. Um die Änderungen

nachverfolgen zu können, wählt **B** die beiden letzten Revisionen aus. **B** wählt aus dem Kontextmenü den Eintrag „Compare revisions“ aus. Aus dem öffnenden Fenster wählt **B** nun die Datei, deren Änderungen verglichen werden sollen (*gedicht.txt*), aus und machen Sie einen Screenshot von dem visuellen Diff (Fenster „TortoiseGitMerge“). Dies wiederholen Sie für die vorletzte und die vorvorletzte Revision und erstellen wieder einen Screenshot.

9. **C** behebt in der Datei *gedicht.txt* den dritten Fehler (Zeile 14 „eierf“), committet die Lösung und pusht sie (nach Person **A** und **B**). Dies erzeugt eine Fehlermeldung, dass es Änderungen im GitHub-Repository gibt, welche **C** nicht hat und welche in Konflikt mit den Änderungen von **C** stehen. **C** folgt den Anweisungen und pullt zuerst und macht einen Screenshot vom Update. Dies erzeugt wieder eine Fehlermeldung, da es einen Konflikt zwischen den Änderungen von **C** und denen von **B** gibt. Was muss **C** tun? **C** klickt auf *Close* und wird gefragt, ob die Änderungen angezeigt werden sollen. **C** klickt auf *Yes*. **C** klickt doppelt auf den Konflikt, um sich das TortoiseGitMerge-Tool anzeigen zu lassen. **C** rechtsklickt auf die rotgefärbte Zeile im Fenster links oben und wählt „use this text block“. Da die Änderungen von **C** jetzt verloren gehen würden, da nicht automatisch gemergt werden kann, muss **C** seine Änderung unten in die grüne Zeile per Hand schreiben und korrigiert „eierf“ wieder. Dann speichert **C** die Änderungen, klickt auf „Mark as resolved“ und schließt das Fenster. Nun muss **C** das gemergte Gedicht comitten und pushen. Vom commit kann **C** wieder einen Screenshot machen.

10. **A, B** und **P** pullen sich die neuste Version des Gedichtes.

11. **P** erzeugt ein Release, indem auf dem Kontextmenü des Ordners seiner Kopie des Repositories den Eintrag „Create Tag…“ gewählt wird. Der Name soll *release_1* sein. **P** kann einen Screenshot der Logs machen.

12. **A**, **B**, und **C** erzeugen je einen Branch. Wählen Sie aus dem Kontextmenü ihres Arbeitsverzeichnisses den Eintrag „Create Branch…“ aus und geben Sie folgende Namen ein:
    A: *Zweig_A*
    B: *Zweig_B*
    C: *Zweig_C*
    Vergessen Sie nicht, den Haken bei „Switch working copy to new branch“ zu setzen, um für die nächste Aufgabe Änderungen im Zweig vorzunehmen. Falls Sie dies vergessen haben, können sie mit „TortoiseGit“ → „Switch/Checkout…“ in den Zweig wechseln.

13. **C** löscht nun die Datei *gedicht.txt* in der Kopie über die Funktion „löschen“ aus dem Explorer-Kontextmenü.

14. **B** löscht nun die Datei *gedicht.txt* über „TortoiseGit → Delete“ und committet die Änderungen.

15. **B** und **C** machen das Löschen rückgängig über „TortoiseGit → Revert“ und wählen das Gedicht aus.

16. **A**, **B** und **C** fügen in ihren Zweigen eine weitere, genau vierzeilige Strophe (aus einem beliebigen anderen Gedicht oder frei erfunden) an das Ende des Gedichtes an. Wichtig

ist dabei, dass sich die Strophen von **A**, **B** und **C** unterscheiden. **A**, **B** und **C** committen und pushen Ihre Änderungen.

17. **P** korrigiert den 4. Fehler (Zeile 20 „serabrednuw“) im Hauptzweig, committet und pusht die Änderungen.

18. **A, B** und **C** übernehmen diese Korrektur in Ihren Zweig (natürlich mit Git!). Gehen Sie bei diesem Schritt besonders konzentriert vor! Wählen Sie auf dem Kontextmenü „TortoiseGit“ → „Pull…“ und im anschließenden Dialog wählen Sie als „remote Branch“ „master“ aus. Belegen Sie das Ergebnis mit einem Screenshot aus des Revision Graphs (Kontextmenü: „TortoiseGit → Revision Graph“).

19. **P** führt alle Zweige wieder zusammen, so dass die Datei *gedicht.txt* dann sieben Strophen umfasst und alle Fehler behoben sind. **P** pullt sich die neuen Branches von **A**, **B** und **C**. Nun mergt **C** den Master-Branch mit *Zweig_A*. Dazu wählt **C** das im Kontextmenü „TortoiseGit → Merge“ aus. Hier wählt **C** bei *From* den Branch *remotes/origin/Zweig_A* aus und bestätigt mit *ok*. Nun mergt **C** *Zweig_B* mit dem Master-Branch. Hierbei tritt ein Konflikt auf. Um diesen zu lösen, klickt **C** auf resolve. Im folgenden Dialog klickt **C** doppelt auf die rote Datei im unteren Bereich des Fensters. Nun klickt **C** doppelt auf die ganzen Fragezeichen im unteren Teilfenster und wählt „use text block from ‚mine‘ before ‚theirs‘. Dies wird gespeichert und der Konflikt als *resolved* markiert. **P** committet den merge. Nun mergt **P** *Zweig_C* genauso wie *Zweig_B* und committet das Ergebnis anschließen wieder. Das Gesamtergebnis wird von **P** gepusht und Sie können wieder einen Screenshot des Logs erzeugen.

20. **P** erzeugt ein Release namens „release_2“. Wie sieht das Delta zu „release_1“ aus? Dazu ruft **P** die Logs auf, wählt dann die Revisionen der beiden Releases aus und wählt im Kontextmenü „Compare revisions“ aus. Im erscheinenden Dialog wählt **P** die Datei *gedicht.txt* aus und erzeugt einen Screenshot vom Fenster vom Diff (Fenster „TortoiseGitMerge“).

21. Sie können auch sehen, welcher Entwickler welche Änderung eingecheckt hat – wählen Sie hierfür den Eintrag „TortoiseGit“ → „Blame…“ aus dem Kontextmenü der Datei *gedicht.txt*. Auch hiervon machen Sie einen Screenshot.

"Weihnachten"

Markt und Straßen stehn verlassen,
Still tethcuelre jedes Haus,
Sinnend geh ich durch die Gassen,
Alles sieht so festlich aus.

An den Fenstern haben Frauen
Buntes Spielzeug fromm geschmückt
Tausend Kindlein stehn und schauen,
Sind so wunderstill beglückt.

Und ich wandre aus den Mauern
Bis suanih ins eierf Feld,
Hehres Glänzen, heilges Schauern!
Wie so weit und still die Welt!

Sterne hoch die Kreise schlingen,
Aus des Schnees Einsamkeit
Steigts wie serabrednuw Singen -
O du gnadenreiche Zeit!

--- gedicht.txt ---