Labordokumentation
Code-Archivierung mit Git

Schule:	Berufsbildende Schule Metalltechnik & Elektrotechnik der Region Hannover
Klasse:	FSEB 15
Unterrichtsfach:	Netzwerke
Thema:	Code-Archivierung
Verfasser:	xxx
Betreuer:	Herr Kai Dorau
Abgabetermin:	09.05.2017

Kontakte der Projektbeteiligten

Auftraggeber:

Herr Kai Dorau
Berufsbildende Schule Metalltechnik Elektrotechnik
Otto-Brenner-Schule
Lavesallee 14
30169 Hannover
dorau@bbs-me.de

Projektteam:

xxx	xxx	xxx

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der Weiterbildung zum „staatlich geprüften Techniker“ in der Fachrichtung Informations- und Kommunikationstechnik an der bbs|me Hannover, wird im zweiten Jahr der Ausbildung ein Laborprojekt im Lerngebiet Netzwerktechnik durchgeführt. Das Thema des Projekts ist, Quellcode von Softwareprojekten oder andere Dokumente über mehrere Entwickler hinaus zu verwalten.

Als zu verwendendes System haben sich die Projektmitglieder für das Versionskontrollsystem **Git** entschieden, welches konfiguriert, getestet und anschließend mit Wireshark hinsichtlich seines Verhaltens im Netzwerk analysiert werden soll.

Git ist ein Versionskontrollsystem (VCS), mit dessen Hilfe Änderungen an Dateien über die Zeit hinweg protokolliert werden können, sodass man jederzeit auf Versionen und Änderungen dieser Dateien zugreifen kann. Ein VCS erlaubt es, einzelne Dateien in einen früheren Zustand zurückzusetzen und nachzuvollziehen, wer welche Änderungen daran vorgenommen hat.

Wireshark ist ein Netzwerk-Analyseprogramm, um den Datenverkehr in einem Netzwerk zu analysieren und zu protokollieren. Dies hilft Administratoren bei der Suche nach Netzwerkfehlern oder Schadsoftware, die sich im Netzwerk verbreiten können.

2 Erwartungshaltung bezüglich der Aufgabenstellung

Zu erwarten ist, dass das Labor gemäß dem Lasten- und Pflichtenheft durchgeführt wird.

- Mit Hilfe von Git wird eine Versionsverwaltung etabliert, die es ermöglicht, mit allen Entwicklern eines oder mehrerer Projekte sinnvoll und zielgerecht zusammenzuarbeiten. Versionen einzelner Entwickler sind nachvollziehbar verwaltet. Versionsstände können zu jedem Zeitpunkt eingesehen oder aktiviert werden.
- Git wird dezentral betrieben und damit ist der Entwickler in der Lage, auch ohne Netzwerkzugang Quellcode oder andere Dokumente zu verwalten.
- Wird Git dezentral ohne Netzwerkzugang betrieben, muss sich das Repository nach Netzwerkzugang mit den anderen Repositories synchronisieren. Andernfalls sind die Versionen nicht einheitlich.
- Quellcode von Projekten ist das Asset vieler Softwareprojekte und müssen entsprechend geschützt werden. Bei der Synchronisation über Netzwerke müssen die Daten verschlüsselt übertragen werden.
- Werden Quellcodes oder Dokumente von mehreren Entwicklern gleichzeitig verändert, werden diese Dateien mittels eines Vereinbarungsprozesses synchronisiert und enthalten abschließend alle Änderungen der Entwickler.
- Der Umgang mit Git wird mit einem Client-Programm gewährleistet, das in unterschiedlicher Form auf allen gängigen Betriebssystemen laufen.
- Die Daten in den Repositories müssen regelmäßig gesichert werden. Dies ist nicht Aufgabe von Git, muss aber separat installiert werden.
- Git-Repositories müssen gepflegt werden und stellen einen zum Teil erheblichen Administrationsaufwand dar, der geleistet werden muss.

3 Theoretische Grundlagen der Aufgabenstellung

3.1 Das Problem mit gemeinsam genutzten Dokumenten

Zur Betrachtung der Problematik ist die folgende Darstellung hilfreich:

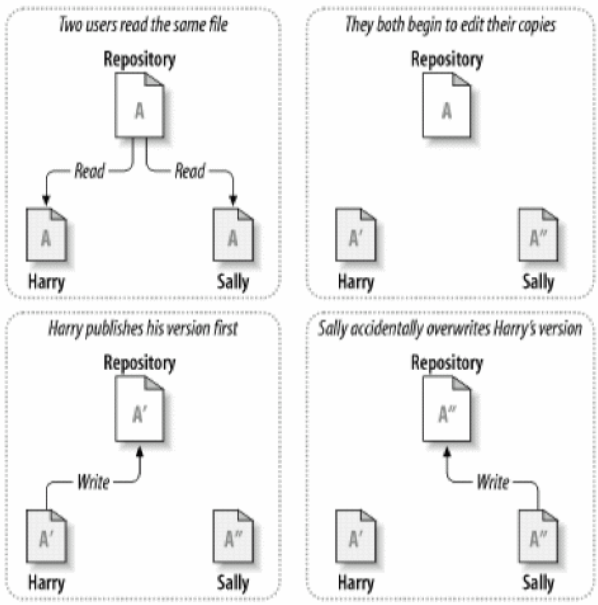

Abbildung 1: Problem gemeinsam genutzter Dokumente

- Zwei Nutzer Harry und Sally öffnen das Dokument A (Abbildung 1 o.l.)
- Harry ändert sein und Sally ihr Dokument. Harry hat nun A‘ und Sally A‘‘ (Abbildung 1 o.r.)
- Harry schreibt seine Änderungen A‘ in A (Abbildung 1 u.l.)
- Sally schreibt ihre Änderungen A‘‘ in A (Abbildung 1 u.r.)

Problem: Sally hat die Änderungen von Harry überschrieben. Damit sind die Daten von Harry nicht mehr vorhanden.

Dieses Szenario ist ein generelles Problem, wenn mehrere Nutzer oder Entwickler an einer Quellcode-Datei arbeiten. **Dieses Problem muss unter allen Umständen verhindert werden!**

3.2 Problemlösung durch Sperren

Ein Dokument kann gesperrt werden, so dass es nur einmal geöffnet werden kann:

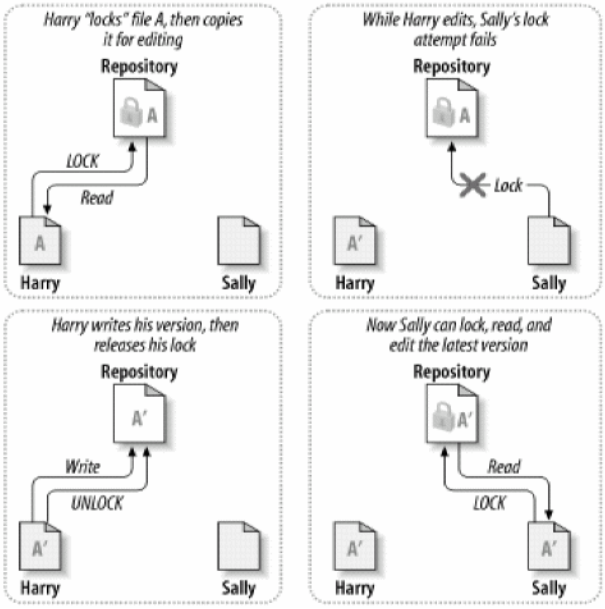

Abbildung 2: Sperren von Dokumenten

- Harry öffnet das Dokument A, sperrt es und arbeitet daran A‘ (Abbildung 2 o.l.)
- Sally versucht, das Dokument A zu öffnen, wird aber gesperrt (Abbildung 2 o.r.)
- Harry schreibt seine Änderungen A‘ in das Dokument A und gibt es wieder frei (Abbildung 2 u.l.)
- Sally kann nun das Dokument A öffnen, um daran zu arbeiten (Abbildung 2 u.r.)

Mit Hilfe dieser einfachen Möglichkeit ist das Problem, das beim mehrmaligen Öffnen eines Dokuments auftreten kann und in 3.1 beschrieben ist, gelöst! Aufgrund von Nachteilen, die diese Möglichkeit mitbringt, ist sie nicht zu empfehlen.

Nachteile:

- Die Freigabe gesperrter Dokumente kann vergessen werden. Andere Nutzer können nicht öffnen und ein späteres Freigeben kostet Zeit.
- Kein gleichzeitiges Arbeiten an einem Dokument möglich.

Vorteil:

- Einfache Lösung ohne Konflikte im Dokument

3.3 Problemlösung durch Mischen

Beim Mischen kann ein Dokument mehrmals geöffnet und bearbeitet werden. Beim Zusammenfügen wird dann ein gemeinsames Dokument erstellt:

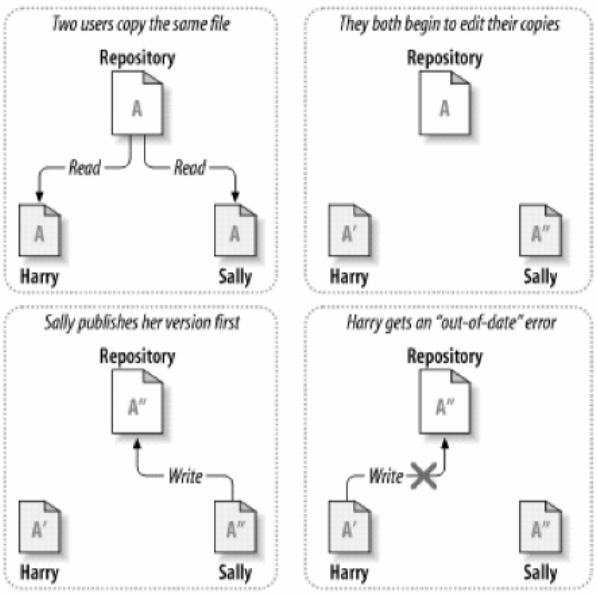

Abbildung 3: Mischen von Dokumenten

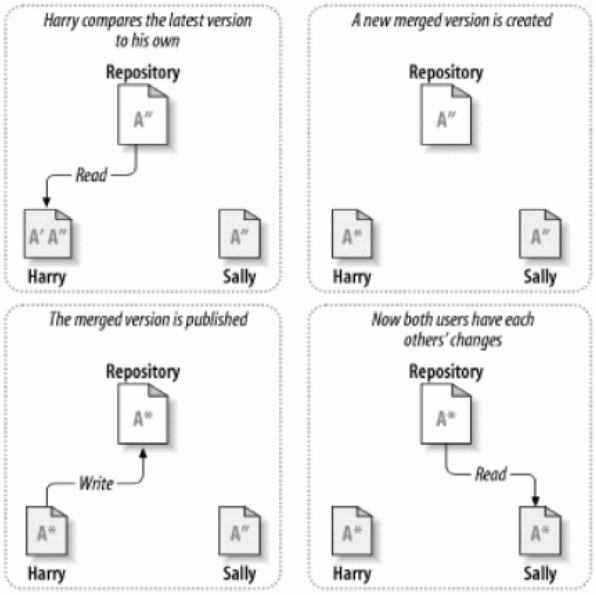

Abbildung 4: Mischen von Dokumenten

- Harry und Sally öffnet das Dokument A getrennt voneinander (Abbildung 3 o.l.)
- Harry ändert das Dokument in A‘, Sally in A‘‘ (Abbildung 3 o.r.)
- Sally speichert ihre Änderungen A‘‘ in A (Abbildung 3 u.l.)
- Harry kann seine Änderungen A‘ nicht mehr in A speichern (Abbildung 3 u.r.)
- Harry vergleicht seine Änderungen A‘ mit der von Sally A‘‘. Beide Versionenwerden gemischt (Abbildung 4 o.l. und o.r.)
- Harry schreibt das neu gemischte Dokument A‘‘ zurück (Abbildung 4 u.l.)
- Sally kann nun das neue Dokument öffnen, um die Änderungen von Harry zu bekommen (Abbildung 4 u.r.)

Das Mischen von Dokumenten wird sehr häufig angewendet. Der Prozess ist i.d.R. problemlos durchzuführen.

Vorteile:

- Das gleichzeitige Arbeiten an einem Dokument ist möglich und auch nützlich
- Dokumente sind nicht gesperrt und können nicht vergessen werden
- Viele Änderungen in einem Dokument führen zu keinem Konflikt, weil sie an unterschiedlichen Stellen im Dokument erstellt werden
- Die Änderungen werden beim Mischen genau protokolliert, sodass sie den Nutzern zugeordnet werden können

Nachteile:

- Bei einem Konflikt sind gleiche Teile des Dokuments von mehreren Nutzern geändert worden. In diesem Fall müssen sich die Nutzer zusammen über das finale Dokument einigen. Ist aber i.d.R. eine Formsache und nicht besonders problematisch.

4 Durchführung und Lösung der Aufgabe

4.1 Git-Branching-Modell

In der Versionskontrolle gibt es eine umfangreiche Auswahl an Branching- und Mergingstrategien. An dieser Stelle sollen lediglich die Grundlagen aufgezeigt werden.
Zuerst wird ein Branch (Zweig) erstellt, da sich der Benutzer zunächst nur im sogenannten Masterbranch befindet. Es ist allgemein zu empfehlen, einen Main-Branch zu erstellen um Fehler zu vermeiden.

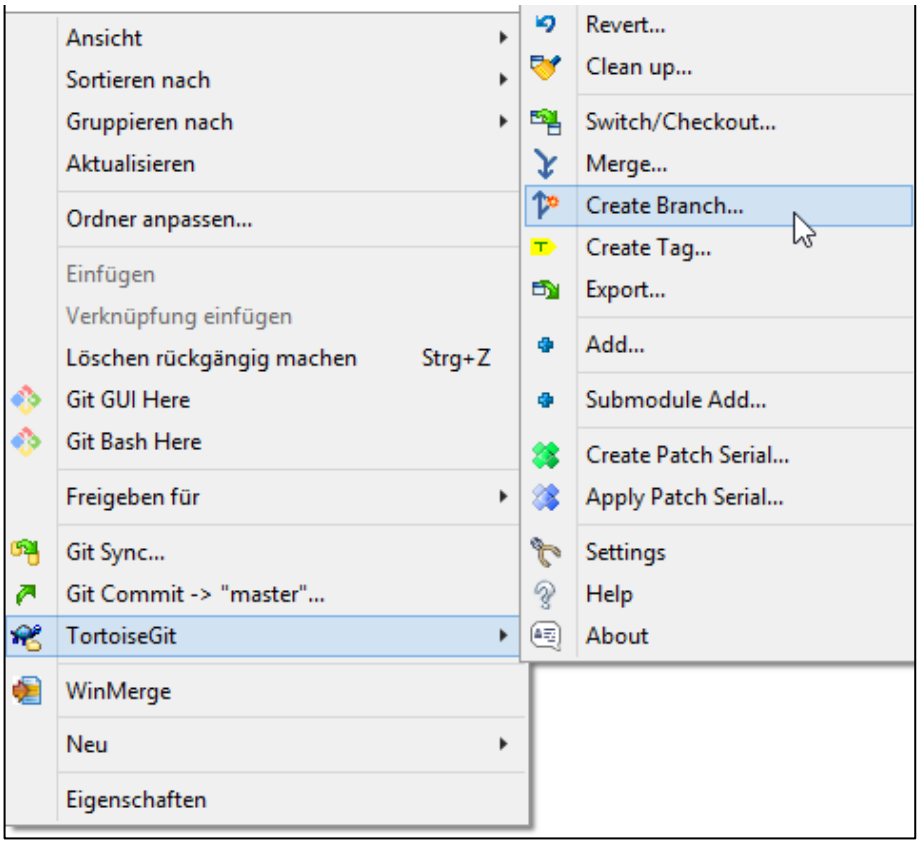

Abbildung 5: Branch erstellen

In diesem Beispiel erstellen wir nun den Branch **Feature_A** und auf die gleiche Weise **Feature_B**. Die Option **Switch to new branch** ist gesetzt, da wir direkt in diesem Branch eine Datei vorbereiten, welche später gemischt (zusammengeführt) werden soll.

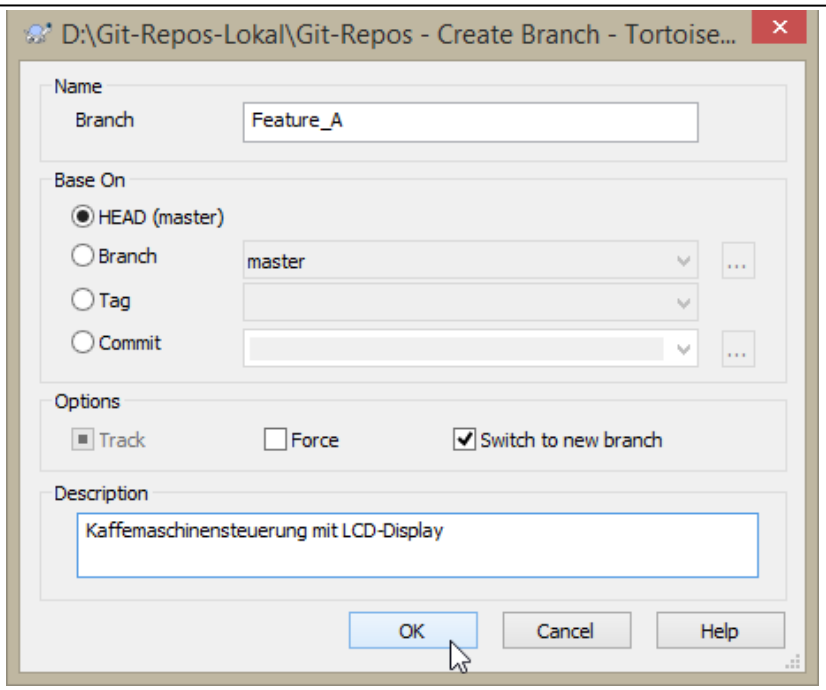

Abbildung 6: Branch benennen und erstellen

Um beide Branches und deren Inhalte zu mischen, wird nun in einem der Ordner per Kontext-menü die Merge-Funktion aufgerufen.

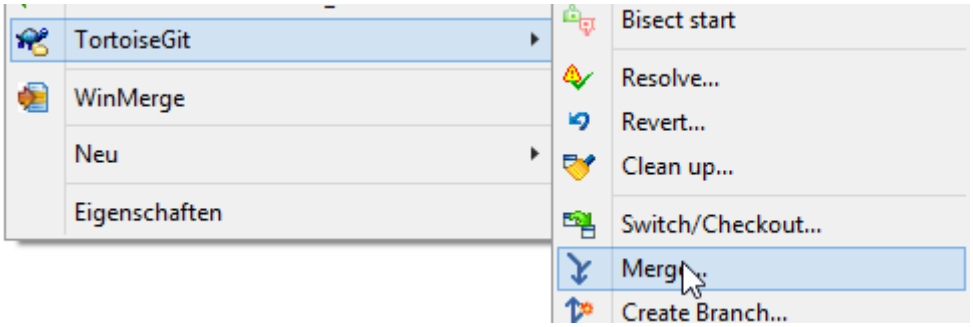

Abbildung 7: Merge-Vorgang starten

Nun muss ausgewählt werden, von welchem Branch man in den aktuellen Branch mischen möchte. In diesem Fall von **Feature_B** in den aktuellen Branch **Feature_A**. Es empfiehlt sich generell immer die Kommentar- bzw. Message-Funktion zu nutzen, um jeden Schritt nachvollziehbar dokumentieren zu können. Diese Funktion kann auch erzwungen werden.

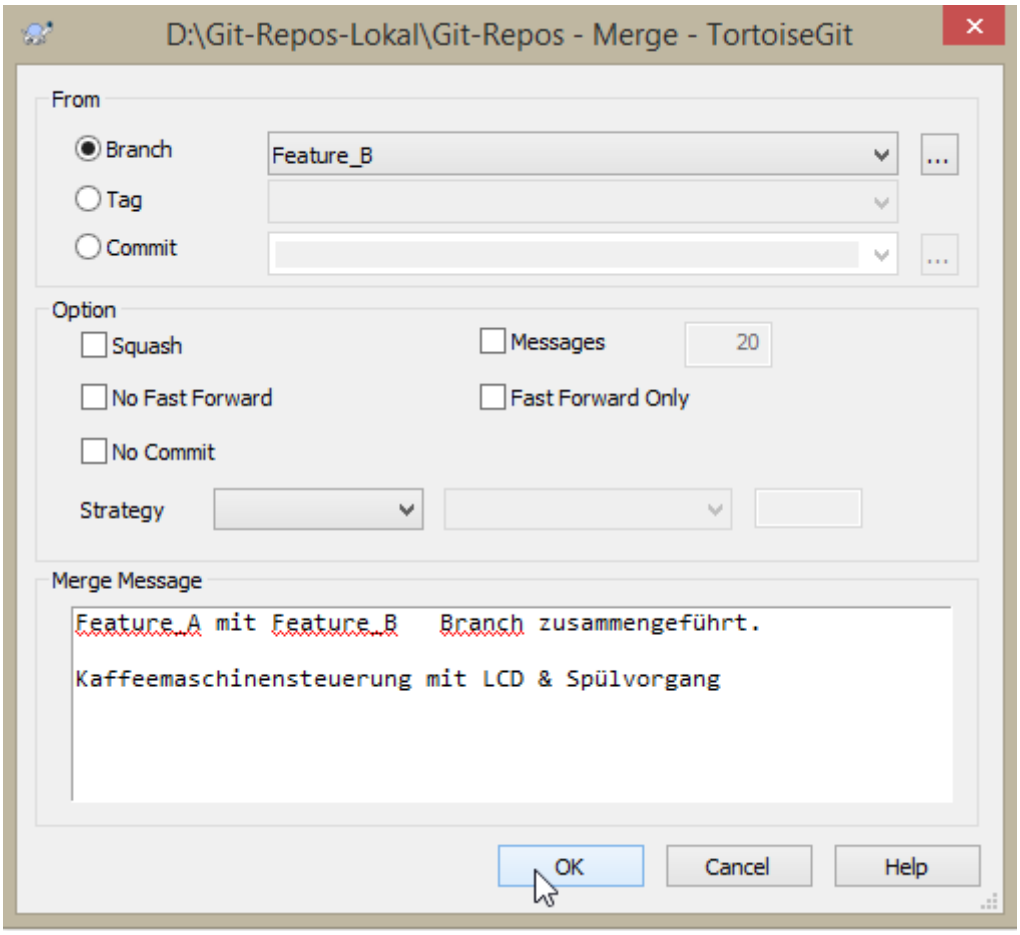

Abbildung 8: Merge-Auswahl

Im folgenden Dialog werden die Konfliktdetails näher beschrieben, welche nach einem **Merge-Vorgang** auftreten können. Man hat die Auswahl verschiedener Optionen. Es kann die Serverversion oder die lokale Version übernommen werden. In diesem Beispiel sollen aber die Programmfunktionen von beiden Branches zusammengeführt werden. Die Differenzen der beiden Konfliktdateien können von Diff- / Merge-Tools angezeigt werden, welche die Zusammenführung erleichtern.

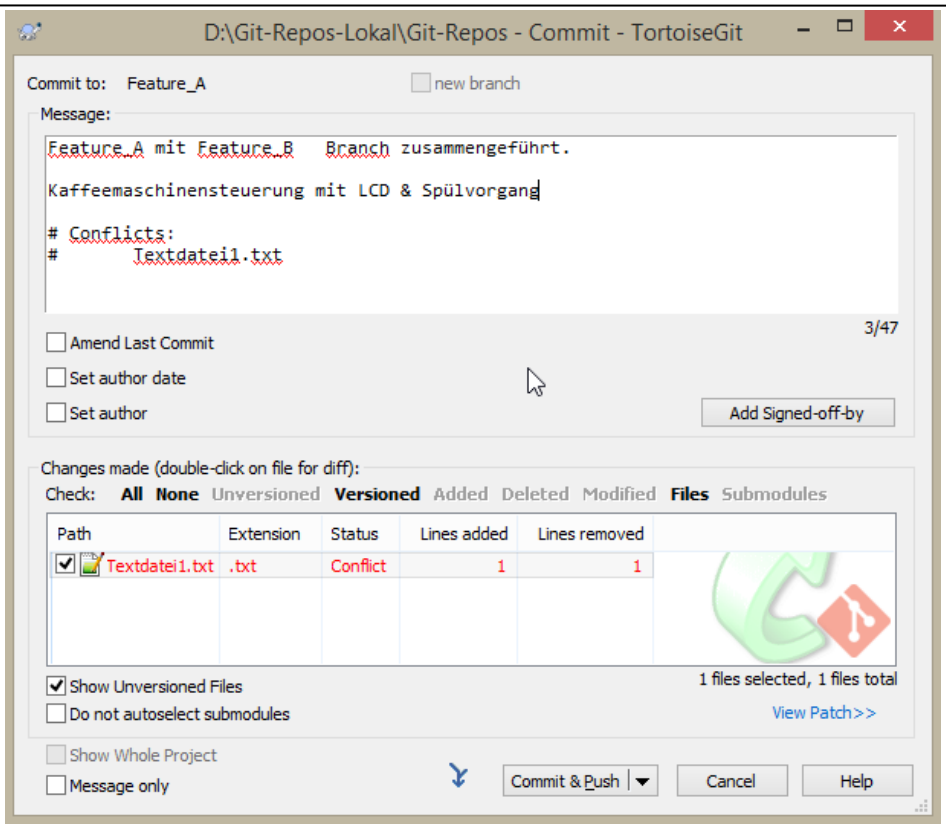

Abbildung 9: Konfliktdetails

Tortoise bietet mit seiner Log-Ansicht außerdem einen guten Überblick der Branch-Topologie.

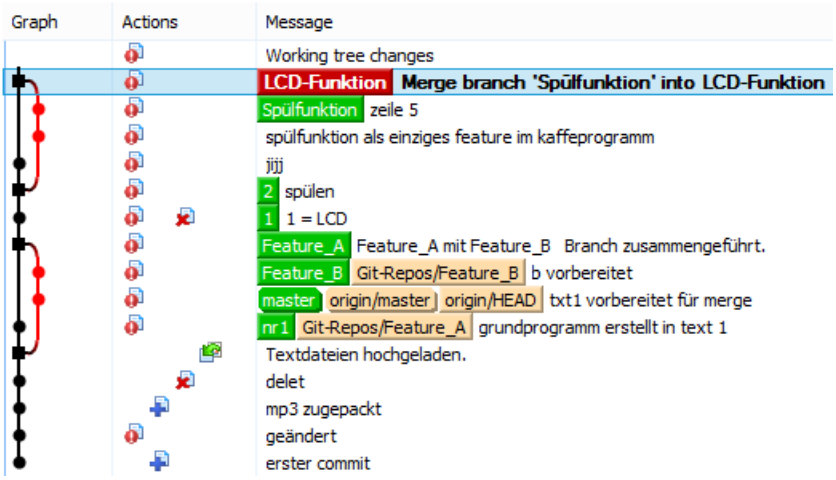

Abbildung 10: Log-Ansicht von Tortoise

4.2 Netzwerkanalyse mit Wireshark

Es sind zwei verschiedene Messungen angesetzt, da es bei Git die Möglichkeit gibt, das SSL-Protokoll und die beim Datenaustausch verwendete Komprimierung der Dateien zu deaktivieren. Bei beiden Versuchen werden die verwendeten Protokolle und Paketgrößen unter Anwendung eines Filters analysiert. Der Filter beschränkt die Anzeige auf die IP-Adressen des Servers und des Clients, um ggf. anderen auftretenden Datenverkehr von der Betrachtung ausschließen zu können. Des Weiteren werden Statistikfunktionen zur quantitativen Darstellung der Protokolle verwendet.

4.2.1 Szenario 1

Standardkonfiguration eines Bare-Repository:
SSL = 1, Kompression = 1, Übertragung mit SSH-Protokoll

config

```
1  #Von Git erstellte Voreinstellungen
2  [core]
3  bare = true
4  repositoryformatversion = 0
5  filemode = false
6  symlinks = false
7  ignorecase = true
```

Abbildung 11: Standartkonfiguration eines Bare-Repository

In dem Log einer Dateiübertragung zum Git-Server werden die „Delta-Compression“ (Differenzen der unterschiedlichen Versionen) und der Hinweis auf eine SSH-Verbindung ausgegeben.

```
git.exe push --progress "origin" master:master

Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 367 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To ssh://192.168.0.70/volume1/Bermudadreieck/Marten/ssl-repo
0237770..a53c300  master -> master

Success (3000 ms @ 09.05.2017 04:57:02)
```

Abbildung 12: Log einer Übertragung zum Server per SSH-Verbindung

Nach dem Mitschnitt eines Push-Vorgangs ist in der Protokollhierarchie aus Wireshark festzustellen, dass der Datenanteil des SSH-Protokolls überwiegt. (aus Sicht des OSI-Schichtenmodells oberhalb der Transportschicht)

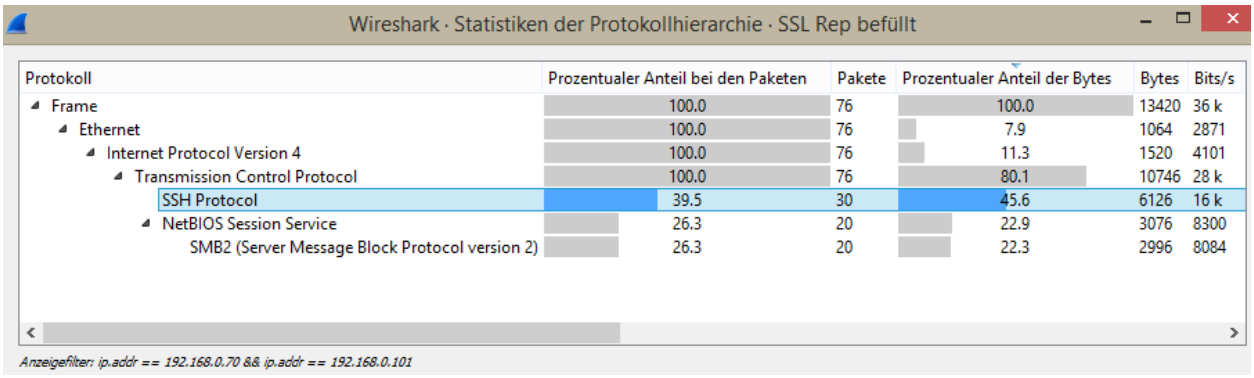

Abbildung 13: Protokollhierarchie der SSH-Verbindung zum Git-Server

In der komplett mitgeschnittenen Datenübertragung zum Git-Server mit einer SSH-Verbindung sind keine Ordnernamen bzw. Dateiinhalte zu identifizieren. Auffällig im Bild ist auch die *sichere* Übermittlung der Daten mittels TCP-Protokoll. Die Benutzung von Port 22 bestätigt die Verwendung des SSH Protokolls (SSHv2).

Source	Destination	Protocol	Length	Info
192.168.0.101	192.168.0.70	SSHv2	93	Client: Protocol (SSH-2.0-TortoiseGitPlink_Release_0.68)
192.168.0.70	192.168.0.101	TCP	60	22 → 55613 [ACK] Seq=1 Ack=40 Win=14656 Len=0
192.168.0.70	192.168.0.101	SSHv2	85	Server: Protocol (SSH-2.0-OpenSSH_6.8p1-hpn14v6)
192.168.0.101	192.168.0.70	SSHv2	1158	Client: Key Exchange Init
192.168.0.70	192.168.0.101	TCP	60	22 → 55613 [ACK] Seq=32 Ack=1144 Win=16832 Len=0
192.168.0.70	192.168.0.101	SSHv2	766	Server: Key Exchange Init
192.168.0.101	192.168.0.70	SSHv2	102	Client: Diffie-Hellman Key Exchange Init
192.168.0.70	192.168.0.101	TCP	60	22 → 55613 [ACK] Seq=744 Ack=1192 Win=16832 Len=0
192.168.0.70	192.168.0.101	SSHv2	678	Server: Diffie-Hellman Key Exchange Reply, New Keys
192.168.0.101	192.168.0.70	SSHv2	70	Client: New Keys
192.168.0.70	192.168.0.101	TCP	60	22 → 55613 [ACK] Seq=1368 Ack=1208 Win=16832 Len=0
192.168.0.101	192.168.0.70	SSHv2	118	Client: Encrypted packet (len=64)

Abbildung 14: Auszug aus Wireshark bei einer SSH-Verbindung

4.2.2 Szenario 2

Konfiguration Szenario 2:

SSL= 0, Kompression = 0, HTTP-Übertragung mithilfe vom WebDAV-Server über Port 5005 = 1

config

```
1   #Von Git erstellte Voreinstellungen
2   [core]
3   bare = true
4   repositoryformatversion = 0
5   filemode = false
6   symlinks = false
7   ignorecase = true
8
9   #SSL-Verifikation deaktiviert
10  [http]
11  sslVerify = false
12
13  #Kompression deaktiviert
14  [pack]
15  # disable delta-based packing
16  depth = 1
17  # disable compression
18  compression = 0
```

Abbildung 15: Konfigurationsdatei der Bare-Repository (Serverseitig)

Der Git-Server muss über das Terminal mit dem Befehl **export GIT_SMART_HTTP=0** konfiguriert werden, unsichere und unverschlüsselte Verbindungen über HTTP zuzulassen. Außerdem muss auf dem serverseitigen Bare-Repository im Ordner **hooks** die Datei **post-update.sample** in **post-update** umbenannt werden. Danach wird im selben Ordner per Git-Bash-Terminal die **post-update** Datei mit dem Execute-Befehl ausgeführt. Zudem muss die anonyme Authentisierung im WebDAV-Server aktiviert sein. Dies ist über die GUI der Diskstation durchzuführen. Im Log einer abgeschlossenen Übertragung mit den oben aufgeführten Einstellungen kann man erkennen, dass keine Komprimierung stattgefunden hat.

```
git.exe push --progress "origin" master:master

Fetching remote heads...
refs/
refs/heads/
refs/tags/
updating 'refs/heads/master'
from 50e60f4cf3bb2ee3ddb2db4c38304207ae880d19
to   bf3206660b39efecf5c09461f1ff8f588517bcb1
sending 2 objects
done
Updating remote server info
To http://192.168.0.70:5005/Bermudadreieck/Marten/no2
50e60f4..bf32066  master -> master

Success (3047 ms @ 09.05.2017 03:36:48)
```

Abbildung 16: Log einer abgeschlossenen Übertragung zum Server ohne Komprimierung

In der Wireshark-Aufzeichnung sind keine SSH-Protokolle ersichtlich. Die Übertragung fand also über HTTP und somit unverschlüsselt statt.

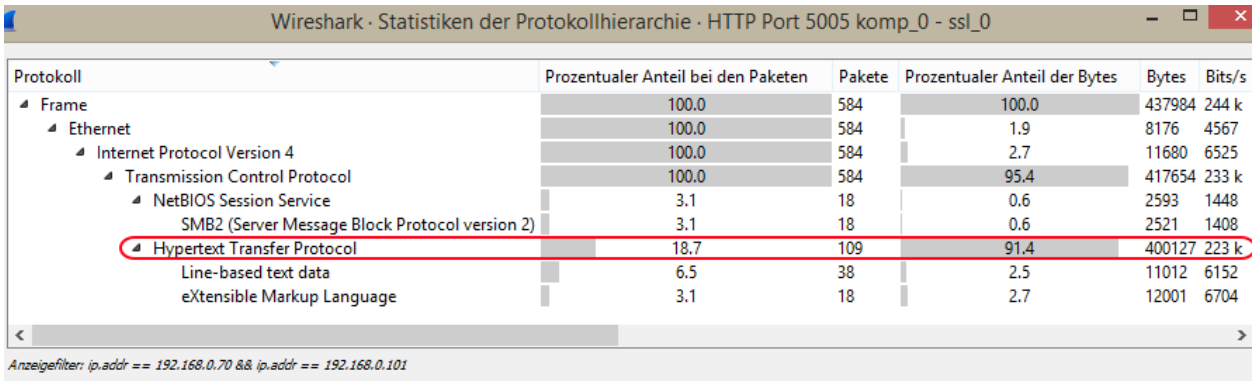

Abbildung 17: Statistikauszug aus Wireshark

In folgendem Bild ist nachgewiesen, dass der Inhalt der Wireshark-Analyse in derselben Form wie in dem Repository wiederzufinden ist. Die Formatierung ist eine Art der Maschinensprache und wurde per HTTP übermittelt.

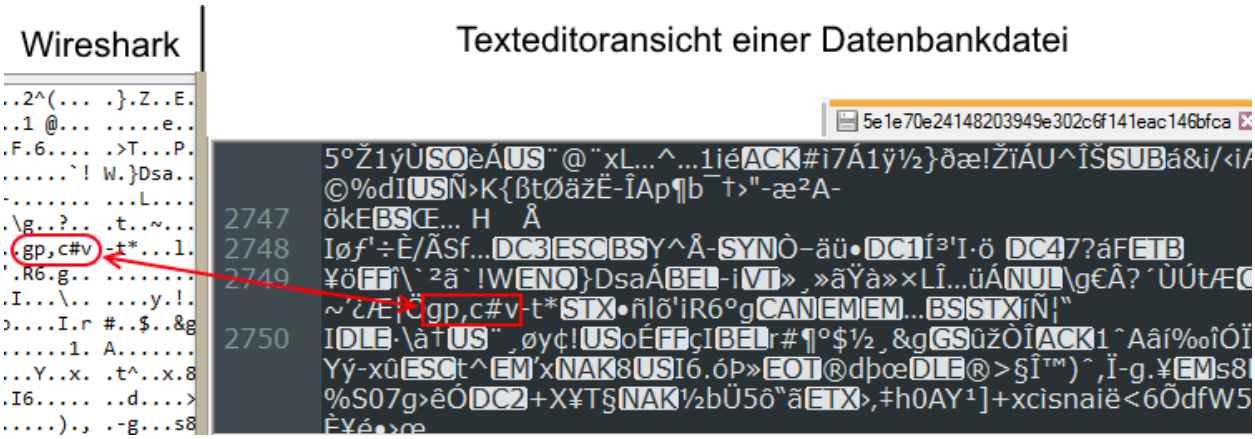

Abbildung 18: Vergleich einer Datenbankdatei mit unsicherer Übertragung

In der Abbildung 17 sind die Klartextbefehle zum Erstellen der Ordnerstrukturen zu erkennen. Im auf der Userseite versteckten Ordner **objects** befinden sich alle Nutzdaten des Repositories.

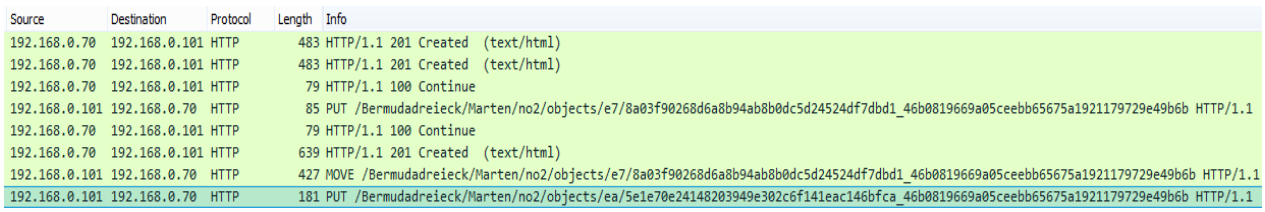

Abbildung 19: Ausschnitt der Daten, mitgeschnitten mit Wireshark

Hier ist der Aufbau einer lokalen Repository zum besseren Verständnis dargestellt. Auch hier ist der aus dem vorigen Bild bereits zu erkennende **ea**-Ordner zu sehen.

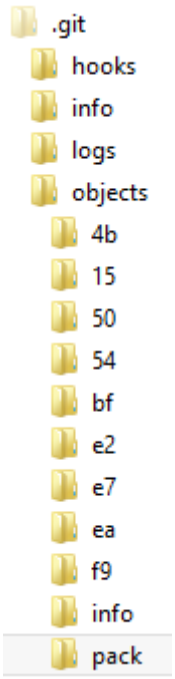

Abbildung 20: Hierarchie einer clientseitigen Repository

5 Fazit und Schlussfolgerung

Im Folgenden wird zunächst die Planung und Durchführung des Projekts reflektiert. Im nächsten Schritt wird dann dem Ergebnis des eigentlichen Projekts Raum gegeben.

5.1 Allgemeine Reflexion der Planung und Durchführung

Bezüglich der Planung und Durchführung ist zu sagen, dass diese im Umgang mit dem VCS Git durch unsere bereits vorhandenen Erfahrungen mit Versionskontrollsystemen nach der vorerst komplizierten Installation problemlos durchgeführt wurde. Die allgemeinen Funktionen konnten durchgeführt und dokumentiert werden.
Die Herausforderung dieser Projektarbeit bestand für uns darin, eine aussagekräftige Netzwerkanalyse mit Wireshark durchzuführen. Für die Durchführung des Projekts musste zunächst einmal ein Server eingerichtet werden. Hier bot sich die Diskstation von Synology an, die bereits vorhanden war. Auf dieser wurde der Git-Server eingerichtet.

5.2 Reflexion der Ergebnisse

In Betrachtung der gewonnenen Erkenntnisse ist eine Arbeitsweise mit dem kostenlosen Versionskontrollsystem Git in Bezug auf sichere Prozessabläufe mit mehreren Nutzern und Schutz der verwendeten Daten im vollen Umfang zu erreichen. Ein ständiger Wechsel der SSH-Keys durch den Diffie-Hellmann Schlüsselaustausch (Abbildung 12) und die sichere Datenübertragung über TCP stellen einen soliden Schutz gegen Manipulation dar und bewahren die Integrität der Daten während des Transfers. Erst die in Szenario 2 genannten Manipulationen machen dieses Versionskontrollsystem transparenter. Auch der Umgang mit Tortoise-Git hat sich im Laufe des Projekts als ein nützliches Werkzeug und eine gute Alternative zur integrierten Konsolenanwendung der Windowsversion von Git hervorgetan. Ob durch die Klartextübertragung des Datenbankcodes eine dritte Person ein komplettes Repository rekonstruieren kann, war in diesem Projekt nicht gefordert und sollte durch die realitätsfernen Umstände auch nicht weiter betrachtet werden.

6 Abbildungsverzeichnis

7 Softwareversionen

Client:

- Git-2.12.2.2-64-bit
- TortoiseGit 2.4.0
- Wireshark 64-bit 2.2.6

Diskstation:

- WebDAV-Server 2.3.3-0024
- Git-Server 2.8.0-0111

8 Abkürzungsverzeichnis/Glossar

Botnet	Gruppe automatisierter Computerschadprogramme
VCS	Version Control System
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
Bare- Repository	beinhaltet nur Dateien und Verzeichnisse im .git Verzeichnis

9 Quellen

- https://www.video2brain.com/de/tutorial/wireshark (09.05.2017)
- https://www.wireshark.org/ (09.05.2017)
- https://wiki.fernuni-hagen.de/helpdesk/images-helpdesk/2/27/Handout.pdf (09.05.2017)
- http://www.webdav.org/ (09.05.2017)
- http://www.f15ijp.com/2012/08/git-ssl-certificate-problem-how-to-turn-off-ssl-validation-for-a-repo/ (09.05.2017)
- https://groups.google.com/forum/#!topic/tortoisegit-users/TmTOXrkD5mw (09.05.2017)
- http://blog.osdev.org/git/2014/02/13/using-git-on-a-synology-nas.html (09.05.2017)
- https://www.video2brain.com/de/git (09.05.2017)
- http://packetlife.net/media/library/13/Wireshark_Display_Filters.pdf (09.05.2017)
- https://www.synology.com/de-de/knowledgebase

10 Anhang

10.1 Allg. Infos zu Git

10.1.1 Repository auf Diskstation erstellen

Serverseitig wird bei der Versionskontrolle mit Git immer eine „Bare-Repository“ erstellt. Dies sagt aus, dass keine Daten für Benutzer erzeugt und dargestellt werden, sondern nur die Metadaten mit den Datenbanken auf dem Server gelagert werden.

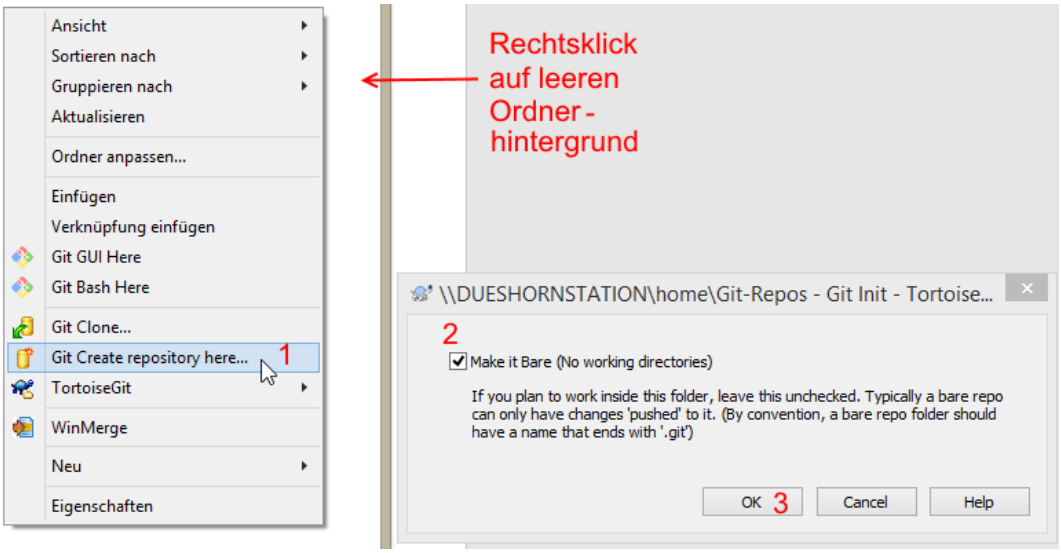

Abbildung 21: Server Repository erstellen (bare)

10.1.2 Git Repository lokal Klonen und Dateien hochladen

Um eine lokale Version der Repository „Git-Repos“ von der Diskstation abzurufen, muss zunächst ein leerer Ordner in Windows erstellt werden. Mit einem Rechtsklick auf die leere Fläche des Ordners wird das Kontextmenü aufgerufen, welches nun mit der Option „Git klonen…“ anbietet, hier eine bestehende Repository zu klonen.

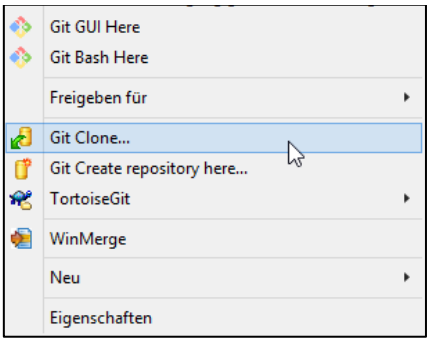

Abbildung 22: Repository Klonen

Im Folgedialog gibt man nun die URL des zuvor erstellten Bare-Repositorys an.

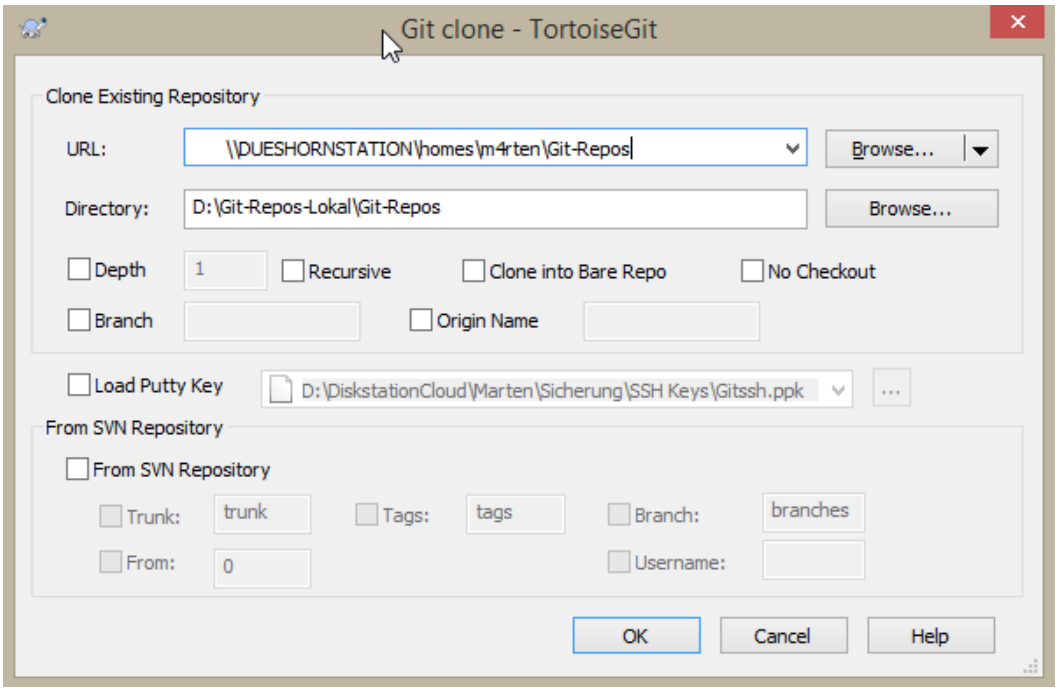

Abbildung 23: Klonvorgangsdialog

Nachdem man den Dialog bestätigt hat, gelangt man zum leeren Ordner zurück. Um diesen nun mit Daten zu befüllen, kopiert man diese in den Ordner. Danach markiert man die Dateien, ruft erneut das Kontextmenü auf und wählt „Add…“ aus.

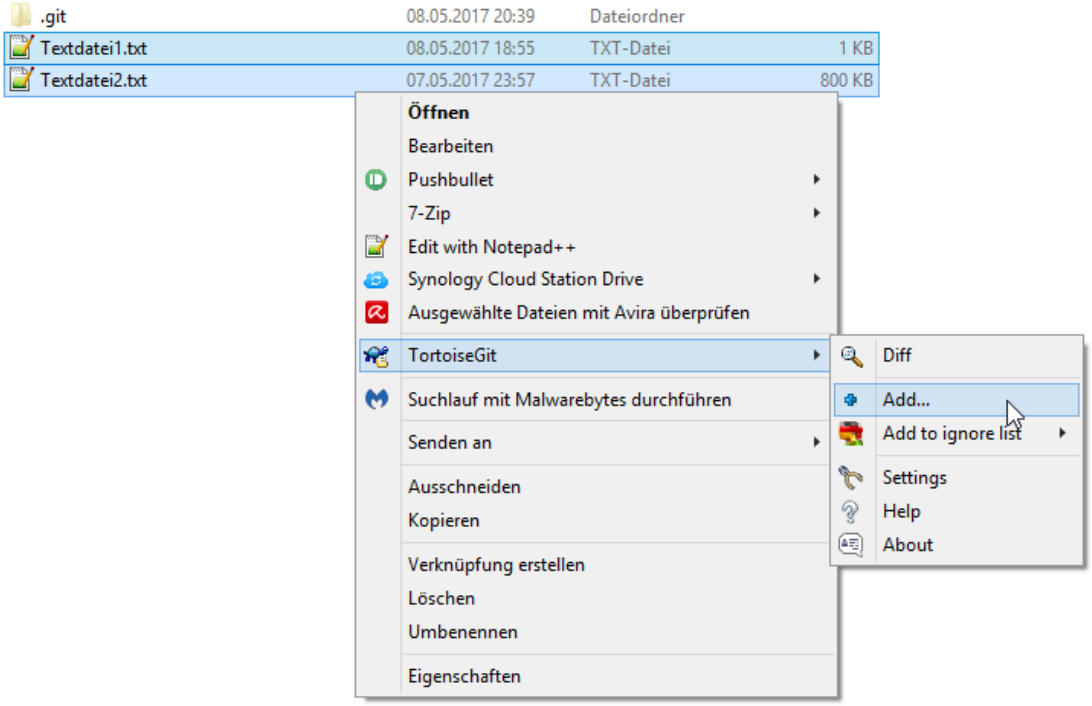

Abbildung 24: Hinzufügen von neuen Dateien

Die Textdateien werden mit dem Add-Vorgang nur „gestaged“. Damit sind diese für den nächsten Commit-Vorgang markiert.

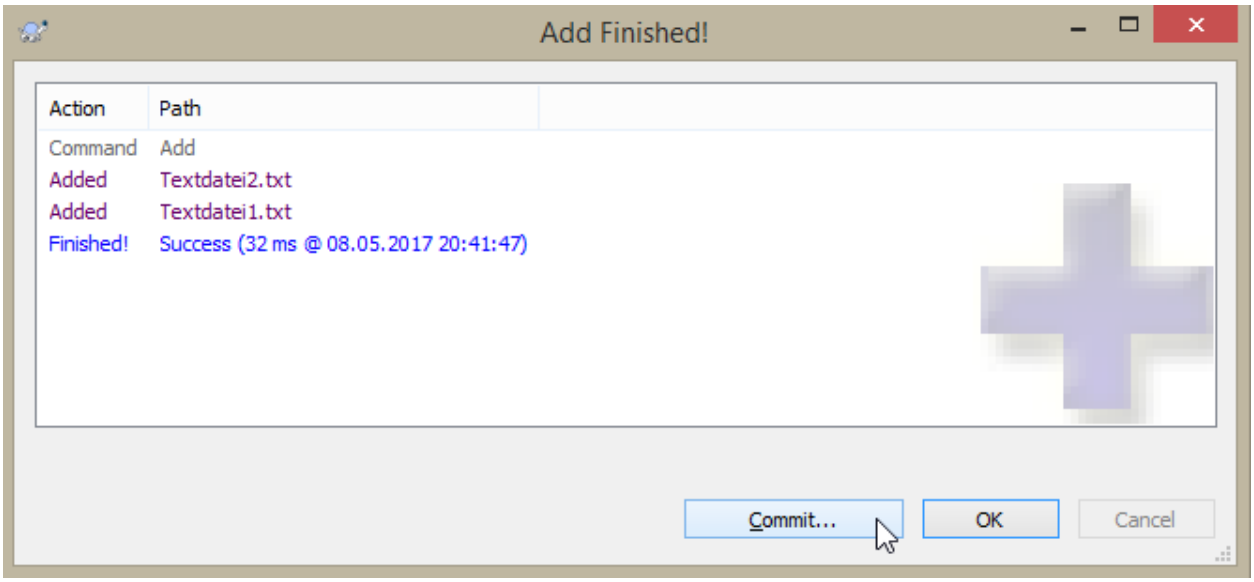

Abbildung 25: Abschlussdialog mit Commit-Option

Mit dem Commit-Befehl werden die Dateien zur lokalen Repository hinzugefügt. Hier wird zusätzlich der Push-Befehl mit ausgeführt, welcher die lokale Repository zur Server-Repository sendet.

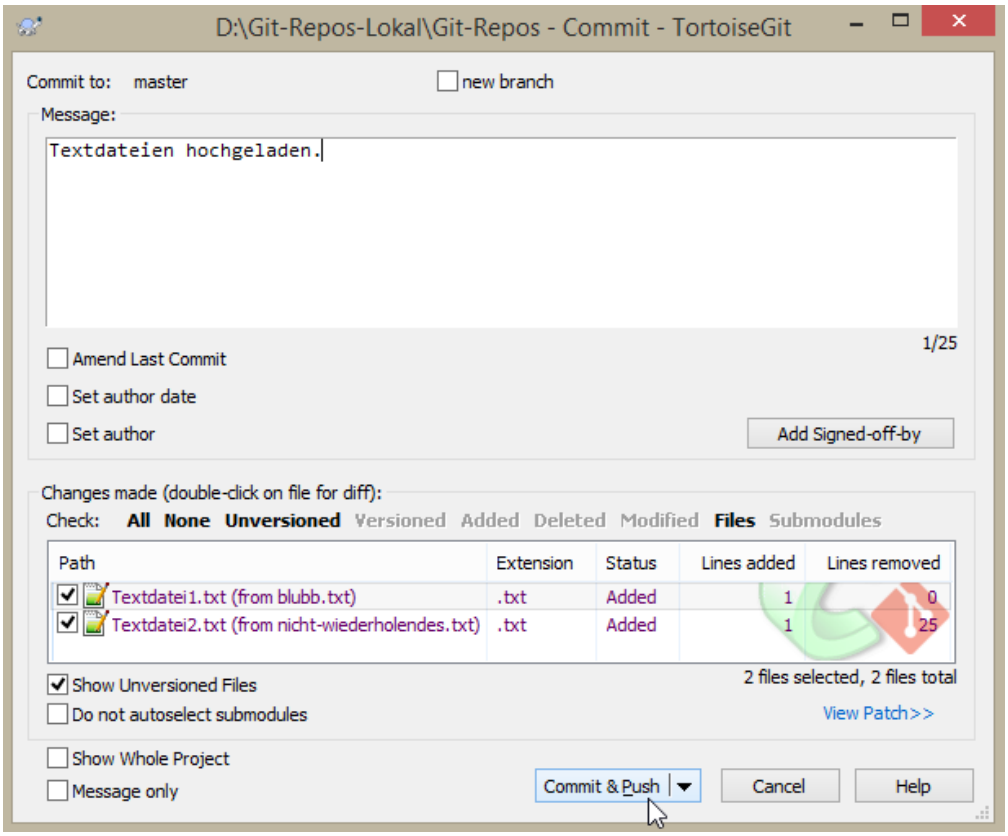

Abbildung 26: Commit& Push Dialog

10.2 Allg. Infos zu Wireshark

Die einzelnen Schichten sind in Wireshark wie folgt dargestellt:

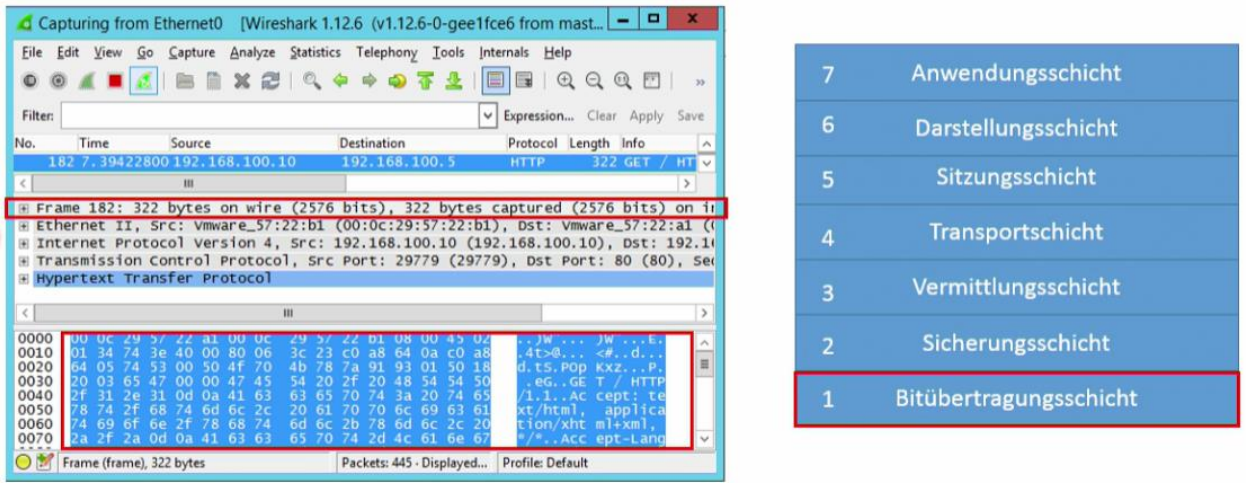

Abbildung 27: Osi-Schichtenmodell in Wireshark (Schicht 1)

In dieser Schicht ist in Wireshark der komplette Datenframe dargestellt. Darin ist die Informationen der Gesamtgröße enthalten, sowie der komplette Frame in Hex dargestellt.

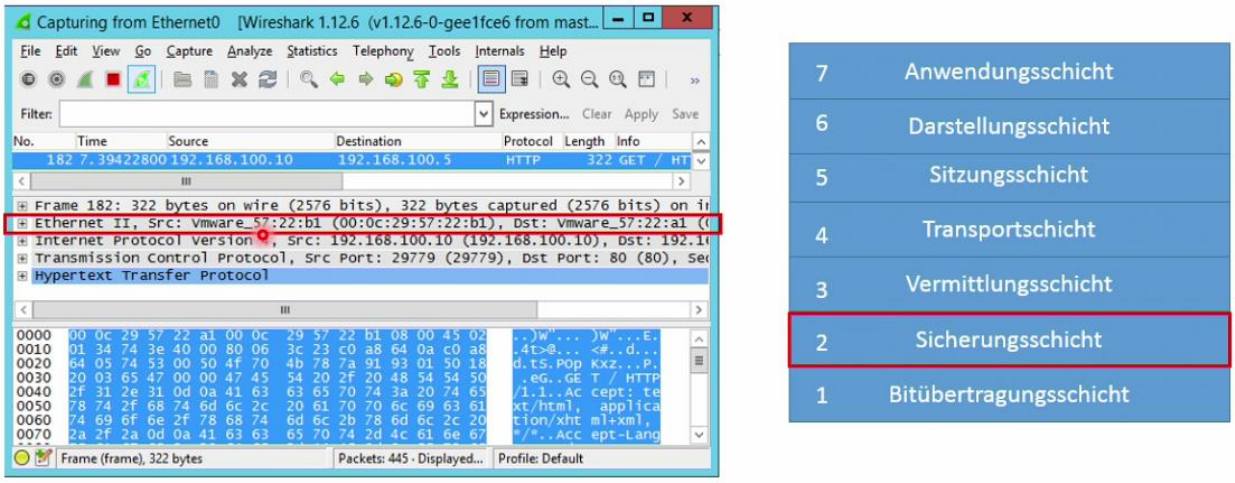

Abbildung 28: Osi-Schichtenmodell in Wireshark (Schicht 2)

In der Sicherungsschicht steckt zB. die Mac-Adresse.

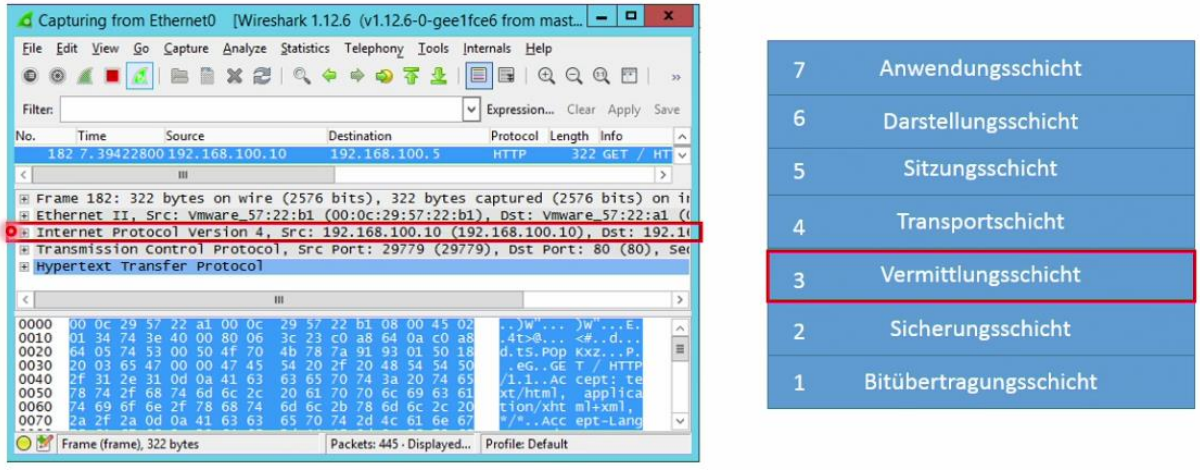

Abbildung 29: Osi-Schichtenmodell in Wireshark (Schicht 3)

In dieser Schicht ist das Internet Protocol zu finden.

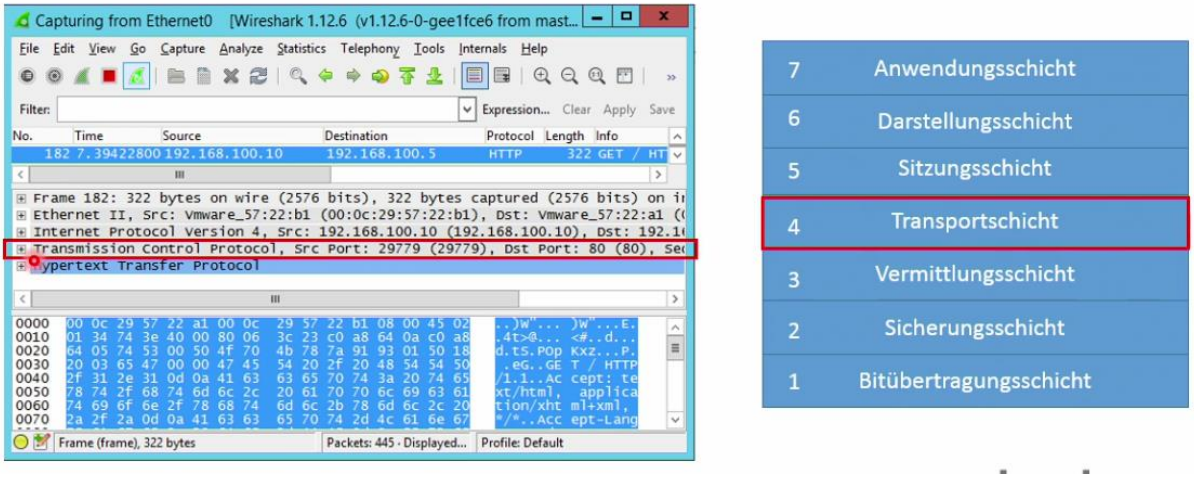

Abbildung 30: Osi-Schichtenmodell in Wireshark (Schicht 4)

In der Transportschicht ist das TCP Protokoll dargestellt.

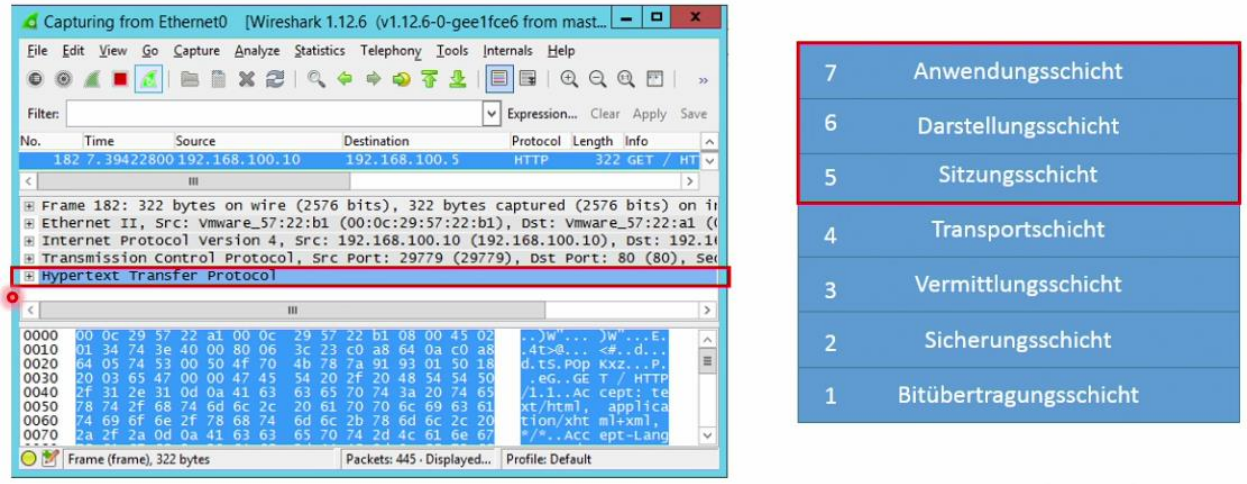

Abbildung 31: Osi-Schichtenmodell in Wireshark (Schicht 5,6,7)

Die letzten 3 Schichten werden im http Protokoll zusammengefasst.

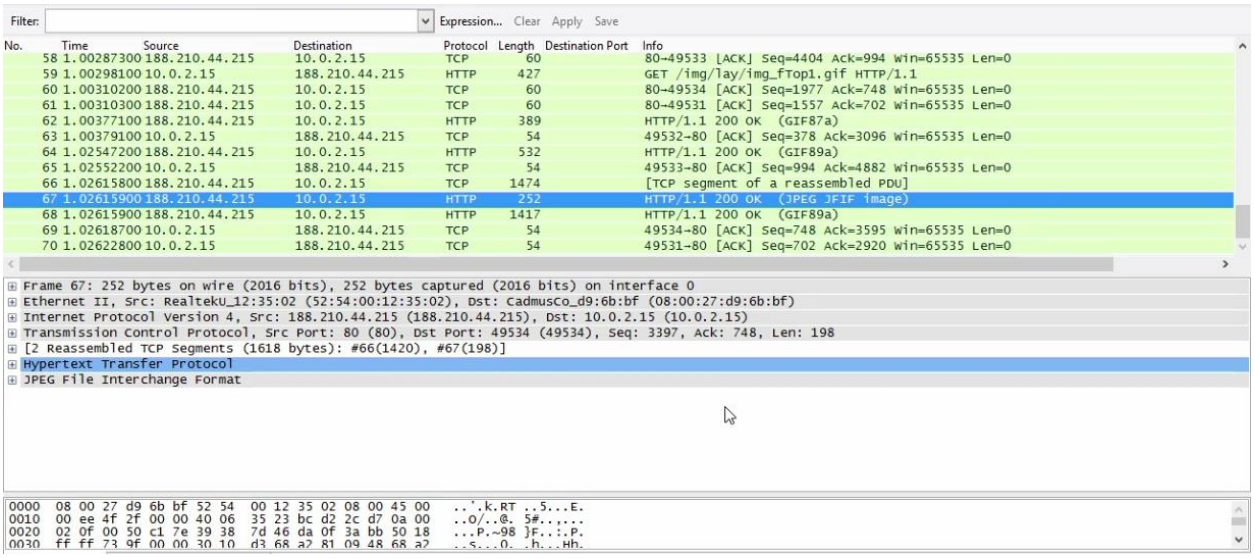

Abbildung 32: Übersicht Wireshark

In der Abbildung 13 ist eine Übersicht der Benutzeroberfläche von Wireshark dargestellt.