Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

Grundlagen Rechnernetze und Verteilte Systeme

IN0010, SoSe 2022

Assignment 0 – Git(Lab) und die VM

Die Veröffentlichung der ersten Programmieraufgabe ist für Montag, den 9. Mai, geplant. Die Aufgaben werden grundsätzlich **nur** über Git veröffentlich und sind nicht über die Weboberfläche einsehbar.

`Assignment 0` ist in GitLab nur zum testen und wird nicht bewertet. Die automatischen Tests überprüfen hier nur, ob ein Programm mit Namen `hello_world` den Text "Hello GRNVS!" ausgibt.

1 Versionsverwaltung (GIT)

Die Abgabe der Programmieraufgaben geschieht über Git. Dabei handelt es sich um eines der gängigen Versionsverwaltungssysteme, welche in der Softwareentwicklung eingesetzt werden. Es ermöglicht mehreren Programmierern gemeinsam an einem Softwareprojekt zu arbeiten. Quelltexte werden in einem **Repository** abgelegt, welches auf einem Server bereitliegt. Von dort kann der Inhalt des Repositories geklont werden, so dass eine lokale Kopie vorliegt. Auf der Kopie kann nun gearbeitet werden. In regelmäßigen Abständen sollte ein **commit** durchgeführt werden, welcher Änderungen in der lokalen Kopie des Repositories abspeichert. Dabei werden im Repository lediglich die Änderungen gegenüber der letzten Revision gespeichert. Um diese Änderungen auf den Server zu übertragen, muss ein **push** ausgeführt werden. Andere Benutzer, wie z.B. die Übungsleitung, können diese Änderungen mittels eines **pull** herunterladen und so ihre lokale Kopie aktualisieren. Ausführliche Informationen über Git finden Sie in [1].

1.1 GitLab

GitLab ist eine populäre Software zum hosten und verwalten von Git Repositories. Neben dem Zugriff über einen Git Client hat es auch eine Weboberfläche (diese eignet sich allerdings nicht zum programmieren). Es stellt zudem zahlreiche zusätzliche Funktionen wie das automatisierte testen des Codes zu Verfügung. Alle Studierende der TUM können das GitLab vom LRZ nutzen (`https://gitlab.lrz.de`). Wenn Sie sich unter `https://grnvs.net/connect` anmelden, werden wir Sie zu den GRNVS Git Repositorien einladen. Sobald die Programmieraufgaben freigeschaltet sind, auch zu den Abgabe Repositories.

1.2 Warum brauchen wir Git?

Es gibt mehrere Gründe, weswegen wir für die Programmieraufgaben Git verwenden:

1. Jeder Informatiker sollte mit wenigstens einem Versionsverwaltungssystem umgehen können. Falls Sie es nicht schon längst können, sollten Sie es besser früher als später lernen.
2. Wir brauchen ein System, mit welchem die Programmieraufgaben abgegeben werden können. Anstatt uns Ihre Programme via Email zu schicken oder als Tarball über ein Webformular hochzuladen, müssen Sie nur sicherstellen, dass Ihre aktuelle Version im Git liegt.
3. Es ermöglicht uns, Ihnen auf elegante Art die Vorlesungsunterlagen bereitzustellen: Sie müssen lediglich die lokale Kopie Ihres material-Gits aktualisieren (ein Befehl oder zwei Klicks) und nicht

jeden Downloadlink einzeln anklicken. Außerdem sehen Sie auf einen Blick, was sich wann geändert hat.

1.3 Woher bekomme ich Git?

- Linux-Nutzer installieren `git` mit der Paketverwaltung ihrer Wahl.
- Nutzer von OS X erhalten `git` entweder über die XCode Kommandozeilen-Tools [2] oder über MacPorts [3] bzw. Brew [4].
- Für Windows-Nutzer empfiehlt sich die Installation von `git` über das *Windows Subsystem for Linux* [5]. Weiter Infromationen finden Sie in Abschnitt 4.
- Falls gewünscht, kann man auch aus einer Vielzahl an grafischen Clients wählen [6].

1.4 Git Repository klonen

Zunächst müssen Sie ein Repository für die einzelnen Programmieraufgaben auschecken. Ihr Repository können Sie in der GRNVS Gruppe `https://gitlab.lrz.de/grnvs/2022ss` unter ihrer persönlichen GRNVS-ID finden (`u<WXYZ>`). Wenn Sie passwortlos arbeiten wollen, richten Sie sich `ssh` (Abschnitt 3.2) ein und achten Sie darauf die SSH URL zu kopieren.

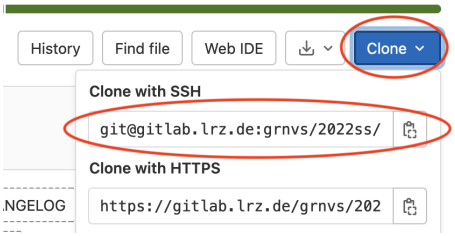


Abbildung 1: Haben Sie Ihr Repository in der Weboberfläche ausgewählt, klicken Sie auf “Clone” und kopieren Sie die entsprechende URL.

1.4.1 Kommandozeile (Linux, OS X und Windows mit WSL)

Die folgenden Befehl klonen die ersten Git Respositories:

```
~ $ git clone git@gitlab.lrz.de:grnvs/2022ss/u<WXYZ>/assignment-0.git <Zielverzeichnis>
~ $ git clone git@gitlab.lrz.de:grnvs/2022ss/material.git <Zielverzeichnis2>
```

1.5 Hilfe zu Git

Für die Verwendung von Git wird auf folgende externe Tutorials verwiesen:

- `https://rogerdudler.github.io/git-guide/index.html`
- `https://try.github.io/`
- `https://git-scm.com/docs/gittutorial`
- `https://xkcd.com/1597/`

2 Die Secure Shell (SSH)

Die `ssh` ermöglicht es auf entfernte Rechner über das Internet zuzugreifen. Der Verbindung ist dabei komplett durch kryptografische Schlüssel abgesichert und verschlüsselt. In der Regel haben Server

kein grafisches Interface, dementsprechend arbeitet man dann komplett auf der Kommandozeile. Zudem vereinfacht `ssh` mit Git zu arbeiten, da man dadurch ohne Passwortanmeldung auf entfernte Git Repositories zugreifen kann.

In der Regel ist `ssh` schon auf Ihrem Rechner vorinstalliert (Linux und WSL), unter MacOS wird es nachinstalliert sobald man es das erste mal aufrufen will. Ansonsten können Sie es analog zu Git in Abschnitt 1.3 installieren.

2.1 SSH Schlüsselpaar generieren

Um auf die VM und Git mit `ssh` zuzugreifen, müssen Sie ein Schlüsselpaar generieren, welches für den Zugriff per SSH geeignet ist. In unserem Beispiel verwenden wir ED25519 Schlüssel.

Sobald Sie ein Schlüsselpaar haben, können Sie den öffentlichen Schlüssel (`<pfad>.pub`) in Ihrem GitLab-Profil eintragen `https://gitlab.lrz.de/-/profile/keys`. Wenn Sie sich dann erneut unter `https://grnvs.net/connect` anmelden, werden wir diesen Key für den Zugriff aud die VMs importieren.

2.1.1 Kommandozeile (Linux, OS X und Windows mit WSL)

Mit diesem Befehl erzeugen Sie ein eigenes Schlüsselpaar:

```
~ $ ssh-keygen -t ed25519
```

Sie werden gefragt, wo der neue SSH-Schlüssel gespeichert werden soll. Standardmäßig wird der private Schlüssel als Datei nach `~/.ssh/id_ed25519` geschrieben. Der öffentliche Schlüssel wird unter `~/.ssh/id_ed25519.pub` abgelegt. Sofern Sie nicht bereits einen SSH-Schlüssel verwenden, brauchen Sie hier nichts zu ändern. Wenn Sie den Befehl ausführen, wird Sie das Programm nach einem Passwort für den neuen privaten Schlüssel und Informationen über Sie fragen. Die Informationen müssen Sie nicht ausfüllen.

2.1.2 GUI für Windows

Wenn Sie unbedingt eine grafische Benutzeroberfäche wollen und den Client *Git for Windows* [7] verwenden, gehen Sie wie in Abbildung 2 gezeigt vor.

3 VServer

Im Rahmen der Programmieraufgaben stellen wir allen zur Vorlesung angemeldeten Studierenden jeweils einen virtuellen „Rootserver / VServer“ (VM) mit öffentlichen IPv4- und IPv6-Adressen zur Verfügung. Die VMs dienen als einheitliche Testplattform, auf denen Ihre Abgaben kompilierbar und ausführbar sein müssen. Der Zugang zu den VMs erfolgt ausschließlich über SSH. Achten Sie deshalb darauf, dass Sie ihren SSH-Key **zuerst** in GitLab eingetragen und sich **danach** unter `https://grnvs.net/connect` angemeldet haben (siehe Abschnitt 1.1).

Da die Bereitstellung von mehr als 1000 VMs eine beträchtliche Menge echter Hardware zur Virtualisierung erfordert, haben wir uns zu folgenden beiden Schritten entschieden:

- VMs müssen einmalig angefordert werden.

- Inaktive VServer werden nach 12 h abgeschaltet (PowerOff[1]) und müssen neu gestartet werden.

Die Anforderung bzw. das Wiedereinschalten von VServern geschieht vollautomatisch und innerhalb weniger Sekunden.

3.1 VM anfordern

Wie eingangs erwähnt müssen Sie Ihre VM zunächst anfordern (bzw. einschalten). Dies geschieht dadurch, dass Sie sich per ssh auf die spezielle VM `grnvs.net` verbinden. Diese wird dann automatisch ihre VM für Sie anlegen und starten.

```
~ $ ssh svm@grnvs.net
```

Dieser SSH-Dienst gibt den Hostnamen der erzeugten VM und die Fingerprints der Hostkeys zurück.

Der Host `grnvs.net` ist über folgende Host Key Fingerprints zu identifizieren:

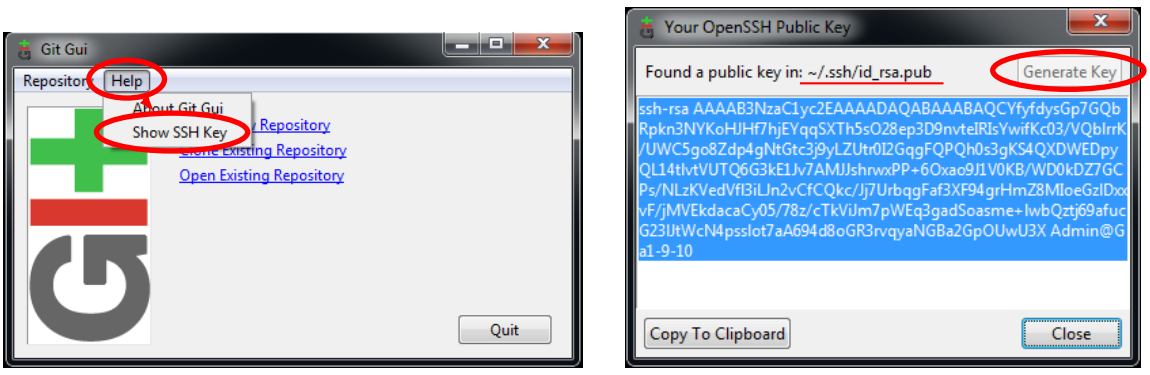
```
256 SHA256:DQ9h/hXw7AppGh5Fi606m9DDE8nNfVxPBWT6Y1rgpRU grnvs.net (ED25519)
3072 SHA256:hvysupUdrPeH3jQTLfyUhVq2d/EgSYpi11jbNDzEVk8 grnvs.net (RSA)
256 MD5:68:68:50:43:34:85:52:97:4b:86:11:2b:e4:73:b9:a1 grnvs.net (ED25519)
3072 MD5:d8:69:0b:bc:3f:4e:8b:51:44:22:55:21:40:0a:49:11 grnvs.net (RSA)
```

3.2 Login auf der VM

Der Benutzername für die VMs ist `root`. Passwort benötigen Sie keines, da die Authentifizierung über ihren SSH-Key erfolgt.

```
~ $ ssh root@svm<wxyz>.net.in.tum.de
```

[1] Es gibt keine Snapshots, d. h. offene nicht gespeicherte Daten gehen verloren.



(a) **(b)**

Abbildung 2: (a) Klicken Sie auf `Help` und dann auf `Show SSH Key`. Es öffnet sich das Fenster in Abbildung 2b. (b) Falls noch kein Schlüsselpaar vorhanden ist, können Sie einen neuen Schlüssel mit `Generate Key` generieren. Wenn schon ein Schlüsselpar vorhanden ist, können Sie hier den Pfad sehen (den werden Sie für SSH später brauchen) und den öffentlichen Schlüssel bequem kopieren.

3.3 Arbeiten auf der VM

Da vermutlich nicht jeder mit den kommandozeilen-basierten Texteditoren wie `vim` oder `nano` umgehen kann oder will, kann im Anschluss das Homeverzeichnis der VM lokal gemountet werden:

- Linux-Nutzer installieren sich SSHFS über ihren bevorzugten Paketmanager.
- OS X-Nutzer haben es leider etwas schwieriger. Aber auch hier gibt es SSHFS-Clients. Eine sinnvolle Anleitung finden Sie unter [8]
- Unter Windows mit WSL2 wird SSHFS wie unter Linux verwendet und installiert. Wird WSL2 nicht verwendet, dann ist es unter Windows komplizierter einen Ordner via SSHFS zu mounten, weshalb es sich unter Umständen anbietet, stattdessen einen Datei-Manager wie WinSCP [9] zu verwenden. (Beschrieben in Abschnitt 3.4.)

  Eine Anleitung zum Einrichten von SSHFS über Windows File System Proxy [10] in Verbindung mit sshfs für Windows [11] folgt.

Unter Linux, OS X und WSL2:

```
~ $ sshfs -o IdentityFile=<pfad> root@svm<wxyz>.net.in.tum.de: <mount-point>
```

Unter Windows ohne WSL2:

- Installieren Sie Windows File System Proxy [10] sowie SSHFS für Windows [11]
- Öffnen Sie die *Systemsteuerung*, wählen Sie *System* und klicken Sie auf *Erweiterte Systemeinstellungen*. Klicken Sie nun auf *Umgebungsvariablen* und fügen Sie den Pfad `%ProgramFiles%\SSHFS-Win\bin` zu der Systemvariablen `Path` hinzu, indem Sie diese auswählen und auf *Bearbeiten* klicken. Entfernen Sie anschließend den Eintrag `%SYSTEMROOT%\System32\OpenSSH\` und bestätigen Sie Ihre Eingabe.
- Nun können Sie in einem neuen `CMD`-Fenster SSHFS wie folgt verwenden:

  ```
  > sshfs -o umask=000 -o IdentityFile=<pfad> root@svm<wxyz>.net.in.tum.de: <mount-point>
  ```

  Dabei ist `<pfad>` absolut anzugeben und `\` doppelt zu verwenden (z.B. `C:\\Users\\Nutzername\\.ssh\\id_rsa` und `<mountpoint>` kann ein Laufwerkbuchstabe sein (z.B. `Z:`)

  Achtung: `ssh` kann nach Änderung der Pfadvariablen nur noch mittels `bash` ausgeführt werden oder Sie übergeben, bei jedem Aufruf, mit dem Parameter `-i <pfad>` den Pfad zum Private-Key.

Alternativ können Sie natürlich auch auf einer lokalen Kopie des Git-Repositories arbeiten. Allerdings müssen Sie es dann jedes mal, wenn Sie es auf der VM testen wollen, zuerst committen und pushen sowie eine zweite Kopie des Repositories auf der VM updaten [12, 13]. Selbstverständlich können Sie auch Ihren eigenen Rechner zur Entwicklung verwenden. Allerdings müssen Sie sicherstellen, dass die Abgabe am Ende auf den VMs lauffähig ist. Außerdem benötigen Sie für die meisten Abgaben Linux. Unter OS X und FreeBSD unterscheiden sich zumindest Name und Pfad einiger Header. Unter Windows wären die Änderungen deutlich umfangreicher, weswegen wir davon abraten.

3.4 WinSCP unter Windows

Stellen Sie zuerst sicher, dass Ihre VM läuft (siehe Abschnitt 3.1). Nun können Sie WinSCP starten.

Als Rechnernamen tragen Sie `root@svm<wxyz>.net.in.tum.de` ein. Dann klicken Sie auf *Erweitert* und im neuen Fenster in der Sektion *SSH* auf *Authentifizierung* und bei *Datei mit privatem Schlüssel* auf „. . . “. Stellen Sie den Filter auf *Alle Dateien* und wählen Sie dann Ihren Private Key aus (normalerweise

in `~/.ssh/id_rsa`). WinSCP wird Sie auffordern, den Schlüssel zu konvertieren. Bestätigen Sie das mit einem OK und speichern Sie den konvertierten Schlüssel ab. Schließen Sie das Dialogfenster und *Speichern* Sie die Konfiguration.

Nun können Sie sich *Anmelden*.

Die restliche Bedienung des Programms sollte relativ intuitiv sein. Interessant ist noch die Funktion *Synchronisieren*, die in der Dokumentation [14] beschrieben ist.

4 Windows Subsystem for Linux (WSL)

Windows 10/11 (Home, Pro und Enterprise) bietet die Möglichkeit, Linux in einer innerhalb von Windows integrierten Umgebung auszuführen. Im Gegensatz zu einer normalen VM hat man jedoch unmittelbaren Zugriff auf das Dateisystem von Windows. Das ermöglicht es, die wichtigsten Linux-Kommandozeilentools unter Windows zu verwenden, z. B. `ssh` und `git`. Beachten Sie bitte, dass die Programmieraufgaben im Allgemeinen aber nicht innerhalb der WSL lösbar sind.

4.1 Installation der WSL

Sie müssen zunächst das Windows Subsystem für Linux installieren. Achten Sie darauf eine aktuelle Windows 10/11 Version zu besitzen. Öffnen sie eine Power-Shell oder Bash-Shell mit Administrator Rechten und installieren Sie das WSL mit dem Befehl:

```
C:\> wsl --install
```

Es empfiehlt sich, danach alle Programme erst mal auf den aktuellen Stand zu holen:

```
~ $ sudo apt-get update
~ $ sudo apt-get upgrade
```

Eine ausführliche Erklärung finden Sie unter [15].

Nach der Installation werden Sie aufgefordert, einen Benutzernamen und ein Kennwort anzugeben. Wir gehen im Folgenden davon aus, dass diese *user* und *password* lauten (bitte wählen Sie etwas sinnvolleres als *password* . . .).

Im Anschluss können Sie die gewählte Linux Distribution (z.B. Debian) über die Windows Suche finden.

4.2 Upgrade auf WSL2

Mit Windows 10 Version 2004 hat Microsoft die Architektur von WSL grundlegend verändert, sodass nun auch FUSE (in unserem Fall notwendig für SSHFS) verwendet werden kann. Es kann sein, dass Sie noch eine alte Version von WSL installiert haben. Folgen Sie in diesem Fall den Schritten unter [16]. Machen Sie sich aber vor dem Wechsel bewusst, dass Sie nach dem Wechsel zu WSL2 weder *VMware* noch *VirtualBox* mehr nutzen können. Wenn Sie virtuelle Maschinen verwenden wollen, dann müssen Sie *Hyper-V* verwenden. Mehr Informationen dazu finden Sie in den WSL2 FAQ [17].

4.3 Überprüfen der Windows Version

Gehen Sie zu **Einstellungen** → **System** → **Info**. Scrollen Sie herunter zum Punkt **Windows-Spezifikation**. Wird dort unter **Version** eine kleinere Zahl als 2004 angezeigt, dann installieren Sie verfügbare Updates oder warten Sie bis ein entsprechendes Update verfügbar ist.

4.3.1 Verhinderung des Wachstums des verbrauchten Arbeitsspeichers

Es kann sein, dass Sie Probleme mit dem Verbrauch des Arbeitsspeichers des WSL2 haben werden. Das WSL 2 alloziert gerne Speicher und gibt diesen nicht wieder frei. Ein Arbeitsspeicherverbrauch von *10 GB* ist hierbei nicht untypisch. Ist dies der Fall, empfiehlt es sich eine Konfigurationsdatei namens `.wslconfig` in `C:\Users\<Name>\` zu erstellen, um den Arbeitsspeicherverbrauch einzudämmen. Der Inhalt könnte wie folgt aussehen:

```
[wsl2]
memory=2GB
swap=0
localhostForwarding=true
```

4.4 Installation von Git

Sie können über den Paketmanager von Debian weitere Pakete nachinstallieren. Dazu benötigen Sie aber root-Rechte, die Sie mittels des Kommandos `sudo su -` und Eingabe Ihres zuvor festgelegten Kennworts *password* erhalten:

```
~ $ sudo su -
~ $ apt update
~ $ apt install git
~ $ exit
```

Der letzte Befehl macht Sie wieder zu Ihrem Standardbenutzer `user`.

4.5 Generieren eines SSH-Keys

Anschließend können Sie einen SSH-Key ähnlich wie in Abschnitt 2.1.1 beschrieben erzeugen. Jedoch müssen Sie beachten, dass Sie **nicht** `ssh-keygen.exe` ausführen.

```
~ $ ssh-keygen -t ed25519
```

5 Anhang: Ein kleines Shell-Einmaleins

`man <Befehlsname>` *manual*
liefert Hilfe zu einem Befehl; funktioniert auch mit Syscalls wie `recv`

`pwd` *print working directory*
gibt das aktuelle Verzeichnis aus

`ls` *list directory contents*
listet die Dateien und Ordner im aktuellen Verzeichnis auf

`cd <Verzeichnisname>` *change directory*
wechselt in ein anderes Verzeichnis, z. B.:

- Arbeitsverzeichnis: `/root/u012`
  Befehl: `cd assignment1`
  wechselt in das Verzeichnis `/root/u012/assignment1`
- Arbeitsverzeichnis: `/root`
  Befehl: `cd /root/u012/result/assignment1`
  wechselt in das Verzeichnis `/root/u012/result/assignment1`

`make`
kompiliert das Programm, wenn sich im aktuellen Verzeichnis eine Datei namens `Makefile` befindet

`cp <Quelle> <Ziel>` *copy*
eine Datei kopieren; zum Kopieren von Ordnern zusätzlich den Parameter „-r“ angeben (`cp -r <Quelle> <Ziel>`)

`mkdir <Ordnername>` *make directory*
einen Ordner erstellen

`rm <Dateiname>` *remove*
eine Datei löschen

`rmdir <Ordnername>` *remove directory*
einen *leeren* Ordner löschen

`mv <Quelle> <Ziel>` *move*
eine Datei oder einen Ordner umbenennen oder verschieben (zum Verschieben einfach bei Ziel einen Pfad mit angeben)

`sudo <Befehl> <Parameter>*` *do as super user*
führt einen Befehl mit den Rechten des Benutzers „root“ aus (andere Betriebssysteme verwenden oft andere Namen wie „super user“ oder „Administrator“); auf der VM haben Sie sich bereits als „root“ angemeldet, also ist der Befehl hier nicht erforderlich

`apt install <Paketname>`
installiert Pakete, also zusätzliche Anwendungen

`./<Programm> <Parameter>*`
führt ein Programm im aktuellen Verzeichnis aus, z. B. `./run -s ../moep8023/moep8023_socket`

`^C` *Tastenkombination: Steuerung+C*
laufendes Programm beenden

`ip n`
gibt die Einträge im ARP-Cache aus

`ip a`
gibt die vorhandenen Netzwerkadapter und ihre Ethernet- und IP-Adressen aus

`host <Rechnername oder IP-Adresse>`
löst via DNS den angegebenen DNS-Namen in eine IP-Adresse auf (*lookup*) oder löst die angegebene IP-Adresse in einen Namen auf (*reverse lookup*)

`tcpdump -w <Dateiname>`
solange `tcpdump` läuft, schneidet es den Netzwerkverkehr mit und speichert ihn in der angegebenen Datei

`exit`
beendet die Sitzung und schließt die Konsole oder trennt die ssh-Verbindung

Ein weiteres nützliches Feature ist Tab Completion: Wenn man gerade einen Pfad eingibt, wie z. B.
`./run ../moe`
kann man die Tab-Taste drücken und erhält als Vorschlag alle möglichen Dateinamen.

Wenn das gerade ausgeführte Programm nicht mehr reagiert, kann man es mit der Tastenkombination `Strg+C` abbrechen.

Literatur

[1] *Git – FAQ*. https://git.wiki.kernel.org/index.php/Git_FAQ.

[2] *XCode CLI-Tools – Download*. https://developer.apple.com/library/ios/technotes/tn2339/_index.html.

[3] T. M. Project, *MacPorts*. https://www.macports.org.

[4] *HomeBrew*. https://brew.sh.

[5] Microsoft, *Frequently Asked Questions about Windows Subsystem for Linux*. https://docs.microsoft.com/en-us/windows/wsl/faq.

[6] *Git – GUI Clients*. https://git-scm.com/downloads/guis/.

[7] *Git – Downloading Package*. https://git-scm.com/download/win.

[8] T. Kessler, *How to mount a remote system as a drive using SSH in OS X*. http://www.macissues.com/2014/10/13/how-to-mount-a-remote-system-as-a-drive-using-ssh-in-os-x/.

[9] *WinSCP*. https://winscp.net/eng/index.php.

[10] *WinFsp – Windows File System Proxy*. https://github.com/billziss-gh/winfsp.

[11] *sshfs for Windows*. https://github.com/billziss-gh/sshfs-win.

[12] *Git – Commit*. https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository.

[13] *Git – Remotes*. https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes.

[14] *WinSCP – Synchronizing*. https://winscp.net/eng/docs/task_synchronize.

[15] Microsoft, *Einrichten einer WSL-Entwicklungsumgebung*. https://docs.microsoft.com/de-de/windows/wsl/setup/environment.

[16] Microsoft, *Upgrade der Version von WSL 1 auf WSL 2*. https://docs.microsoft.com/de-de/windows/wsl/install#upgrade-version-from-wsl-1-to-wsl-2.

[17] Microsoft, *WSL2 FAQ*. https://docs.microsoft.com/en-us/windows/wsl/wsl2-faq.