Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

Grundlagen Rechnernetze und Verteilte Systeme

IN0010, SoSe 2023

Assignment 0 – Git(Lab) und die VM

Die Veröffentlichung der ersten Programmieraufgabe ist für Dienstag, den 2. Mai, geplant. Die Aufgaben werden grundsätzlich **nur** über Git veröffentlicht und sind nicht über die Weboberfläche einsehbar.

1 Setup

In GRNVS verwenden wir das LRZ Gitlab um die Vorlesungunterlagen zu veröffentlichen und die Programmieraufgaben abzugeben und zu testen. Die Aufgaben werden dabei auf virtuellen Maschinen (VMs) bearbeitet, die wir zur Verfügung stellen. Gehen Sie verantwortungsvoll damit um, Sie haben darauf volle Admin Rechte und eine eigene statische IP Adresse. Befolgen Sie dazu die folgenden Schritte:

1. Melden Sie sich im LRZ Gitlab an (→ `https://gitlab.lrz.de`).
2. Laden Sie ihren eigenen SSH Schlüssel im LRZ Gitlab hoch (→ `https://gitlab.lrz.de/-/profile/keys`). Weiter Informationen zu SSH finden Sie in Abschnitt 3.
3. Verbinden Sie GRNVS mit dem LRZ Gitlab (→ `https://grnvs.net/connect`).
4. Sie werden automatisch zu den GRNVS Git Repositories eingeladen. Eines für die Vorlesungsunterlagen, und evtl. weitere für die Programmieraufgaben. Bis zur Veröffentlichung von Assignment 1 gibt es ein Assignment 0, das aber nur dazu dient mit Git experimentieren zu können. Sie finden die Repositories unter `https://gitlab.lrz.de/grnvs/2023ss/`. Weitere Informationen zu Git finden Sie in Abschnitt 2.
5. Wir importieren Ihre SSH Schlüssel von Gitlab um Sie für die VMs zu authentifizieren. Das startet Ihrer eigenen VM geschieht mit folgendem Kommando:

```
0 ssh svm@grnvs.net
```

   Weiter Informationen zu Ihrer VM finden Sie in Abschnitt 4.

Achtung: Wann immer Sie ihre SSH Schlüssel im LRZ Gitlab ändern oder neue Programmieraufgaben veröffentlicht werden, müssen Sie sich erneut unter `https://grnvs.net/connect` anmelden!

2 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].

2.1 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 kaum 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`).

Wir verwenden Gitlab mittels SSH. (Weitere Infos zu SSH in Abschnitt 3). Laden Sie dazu den "offentlichen Schlüssel" (`<pfad>.pub`) in Ihrem GitLab-Profil hoch `https://gitlab.lrz.de/-/profile/keys`. Jedes mal wenn Sie sich unter `https://grnvs.net/connect` anmelden, werden wir Ihre Keys für den Zugriff auf die VMs importieren.

2.2 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.

2.3 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 Informationen finden Sie in Abschnitt 5.
- Falls gewünscht, kann man auch aus einer Vielzahl an grafischen Clients wählen [6].

2.4 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/2023ss` unter ihrer persönlichen GRNVS-ID finden (`u<WXYZ>`). Wenn Sie passwortlos arbeiten wollen, richten Sie sich `ssh` (Abschnitt 4.2) ein und achten Sie darauf die SSH URL zu kopieren.

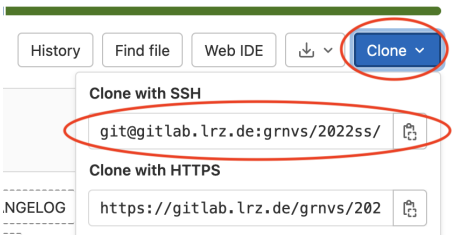

Abbildung 1: Haben Sie Ihr Repository in der Weboberfläche ausgewählt, klicken Sie auf "Clone" und kopieren Sie die entsprechende URL.

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

Die folgenden Befehl klonen Ihre ersten Git Respositories:

```
0 git clone git@gitlab.lrz.de:grnvs/2023ss/u<WXYZ>/assignment-0.git <Zielverzeichnis>
1 git clone git@gitlab.lrz.de:grnvs/2023ss/material.git <Zielverzeichnis2>
```

2.5 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/`

3 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 [5]), unter MacOS wird es nachinstalliert sobald man es das erste mal aufrufen will. Ansonsten können Sie es analog zu Git in Abschnitt 2.3 installieren.

3.1 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.

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

Mit diesem Befehl erzeugen Sie ein eigenes Schlüsselpaar:

```
0 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.

4 Virtuelle Maschinen (VMs)

Im Rahmen der Programmieraufgaben stellen wir allen zur Vorlesung angemeldeten Studierenden jeweils einen virtuellen "Rootserver / Virtuelle Maschiene" (VM) mit öffentlichen IPv4- und IPv6-Adressen zur Verfügung. Die VMs dienen als einheitliche Programmierplattform, 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 2.1).

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 VMs werden nach 12 h abgeschaltet (Es gibt keine Snapshots, d. h. offene nicht gespeicherte Daten gehen verloren.) und müssen neu gestartet werden.

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

4.1 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.

```
0 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:

```
0 2048 SHA256:LWIF8tVvBnooxoQzNausj7im/R6qBcsnQ8MC5ZLhyKE grnvs.net (RSA)
1 256 SHA256:E9cXlVHsqDb85AQBBhUn718eK6D0UxG+baP5hDnJjTY grnvs.net (ED25519)
```

4.2 Login auf der VM

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

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

4.3 Arbeiten auf der VM

Die VMs haben kein grafisches Interface, daher arbeitet man nur über die Kommandozeile. Nützliche Befehle sind in Anhang 6 erklärt.

Es gibt verschiedene Möglichkeiten, wie Sie die VM nutzen können. Sie können Ihr Git-Repository auf der VM klonen, und mit Tools wie `vim` die Hausaufgabe bearbeiten. Falls Sie lieber lokal programmieren und die VM nur zum Testen nutzen, können Sie über Ihr Git-Repository oder mit `rsync` Ihre Änderungen auf die VM kopieren. Viele IDEs können auch beim Programmieren auf einem via SSH zugänglichen Rechner unterstützen. Mehr Informationen zu IDEs finden Sie in Abschnitt 4.4 und im Anhang.

Ein Beispielablauf, wie man mit `rsync` Dateien auf die VM kopiert, wäre wie folgt:

1. Lokal das Assignment im Ordner `assignment-1` bearbeiten
2. das Assignment auf die VM kopieren mit

```
rsync -a assignment-1 root@svmXXXX.net.in.tum.de:/root/assignment-1
```

3. Jetzt können wir uns auf die VM einloggen und dort das Assignment bauen und ausführen

```
ssh root@svmXXXX.net.in.tum.de
cd assignment-1
make
./assignment-1
```

4. Wenn alles geklappt hat, können wir lokal den Code comitten und abgeben

```
git add .
git commit -m"Assignment 1 Submission"
git push
```

4.4 Eine IDE mit Remote Interpreter verwenden

Da vermutlich nicht jeder mit den kommandozeilen-basierten Texteditoren wie `vim` oder `nano` umgehen kann oder will, unterstützen moderne IDEs hybride Entwicklungsprozesse durch "Remote Interpreter". In diesem Fall übernimmt die IDE Teile von dem kommandozeilen-basierten arbeiten. Die genauere Funktionsweise ist aber Programmiersprachen und IDE abhängig. Die Prinzipien von SSH und Git bleiben allerdings die Selben und Sie sollten sie verstanden haben.

Wir können zwei IDEs empfehlen:

- Visual Studio Code (kostenlos und open-source), nähere Informationen in Anhang 7.1
- JetBrains PyCharm Professional (proprietär, aber für Studenten kostenlos), weitere Informationen in Anhang 7.2

5 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.

5.1 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 [5].

6 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/u0012`
  Befehl: `cd assignment1`
  wechselt in das Verzeichnis `/root/u0012/assignment1`
- Arbeitsverzeichnis: `/root`
  Befehl: `cd /root/u0012/result/assignment1`
  wechselt in das Verzeichnis `/root/u0012/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>`)

`rsync -a <Quelle> svmXXXX.net.in.tum.de:<Ziel>`
das Tool kann genutzt werden um einen lokalen Ordner auf die VM zu kopieren (und umgekehrt).

`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.

7 Anhang: Remote Interpreter

7.1 Visual Studio Code

Ausführliche Informationen finden Sie unter “Visual Studio Code Remote - SSH” [7].

Eine Möglichkeit um bequem auf der VM arbeiten zu können, ist mit der Remote-SSH Extension für Virtual Studio Code.

Mittels der in Visual Studio Code eingebauten Dateinavigation, dem Texteditor, sowie dem Terminal können Sie direkt auf der VM die Hausaufgabe herunterladen, bearbeiten, kompilieren/testen und abgeben.

Hierfür gehen Sie wie folgt vor:

1. Stellen Sie zunächst sicher, dass die SSH-Verbindung zur VM eingerichtet ist, dh. dass `ssh root@svm<wxyz>.net.in.tum.de` funktioniert. Die Details dazu wurden bereits in den oberen Abschnitten beschrieben.
2. Nun richten wir VS Code und die Remote-SSH Extension ein:
   - Installieren Sie bei Ihnen Visual Studio Code. Es ist für alle gängigen Betriebssysteme verfügbar (Windows, MacOS, Linux).
   - Starten Sie Visual Stuio Code und öffnen Sie links die *Extension*-Ansicht
   - Suchen Sie nach der „Remote - SSH“-Extension von Microsoft. Installieren Sie diese.

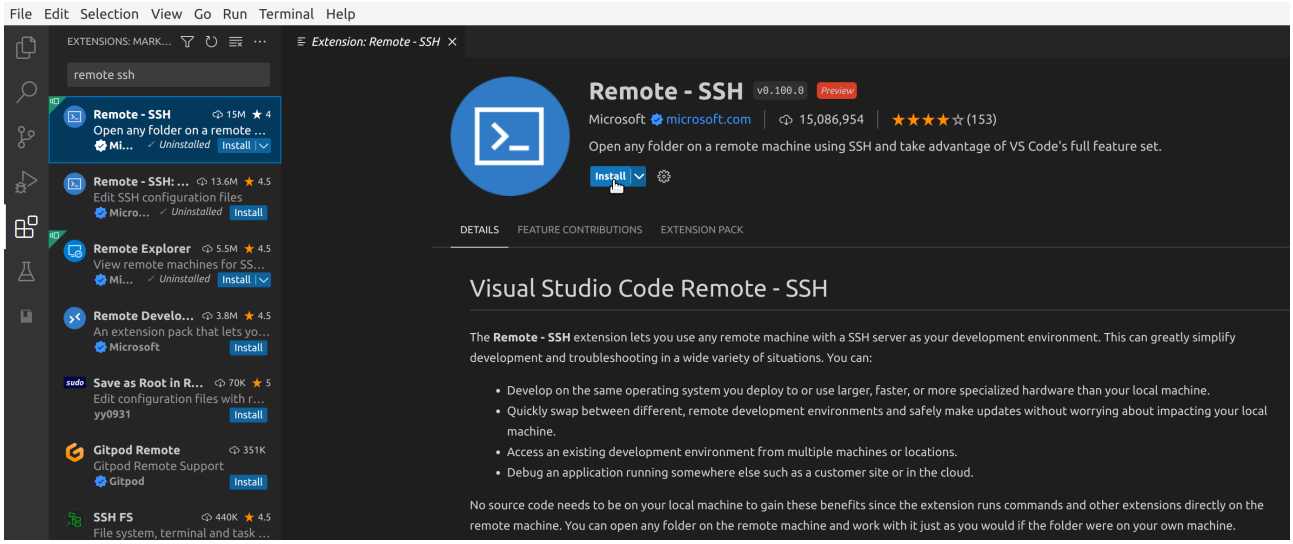

Abbildung 2: Installieren der Remote SSH Extension

3. Jetzt wird eine SSH-Verbindung hinzugefügt. Diese wird in `~/.ssh/config` abgespeichert, daher ist dieser Schritt nur beim ersten Mal nötig:
   - Drücken Sie `F1` und geben Sie in das Suchfeld „remote ssh“ ein.
   - Wählen Sie `Remote-SSH: Connect to Host...` aus.
   - Wählen Sie `+ Add New SSH Host...` aus.
   - Geben Sie `ssh root@svm<wxyz>.net.in.tum.de` ein.
   - Es kann passieren, dass VS Code die Art der VM nicht automatisch erkennt. In diesem Fall wählen Sie die Option Linux.
4. Nun können wir VS Code mit der VM verbinden:
   - Drücken Sie erneut `F1` und wählen Sie den `Remote-SSH: Connect to Host...`-Befehl aus.
   - Nun sollte die VM `svm<wxyz>.net.in.tum.de` in der Liste erscheinen. Wählen Sie diese aus.
   - VS Code öffnet ein neues Fenster, mit dem Sie nun auf der VM arbeiten können.

Sie können sicherstellen, ob alles funktioniert hat, indem Sie links unten auf die grüne Box achten, diese sollte svm<wxyz>.in.tum.de anzeigen. Wenn Sie im Menü unter Terminal | New Terminal ein neues Terminal öffnen, sollte dort auch `root@svm<wxyz>.in.tum.de:~$` stehen.

Wenn Sie an einem späteren Zeitpunkt weiterarbeiten möchten, müssen Sie ggf. die VM mit `ssh svm@grnvs.net` neu starten.

7.2 PyCharm und Python

Im folgenden zeigen wir das Einrichten eines Remote Interpreters für JetBrains PyCharm und Python. Diese IDE können Sie als Student kostenlos nutzen. Achten Sie darauf, dass Sie die Pycharm Professional Edition nutzen. Das Einrichten eines Remote Interpreters ist in der Pycharm Community Edition nicht verfügbar. Die IDE übernimmt dabei das kopieren der Dateien auf die VM und das ausführen des Programms. Man kann auch den visuellen Debugger nutzen. Genauere Informationen gibt es online [8].

Unten rechts kann man den genutzten Interpreter ändern und auch neue hinzufügen. Wählen Sie hier den SSH Interpreter aus wie in Abbildung 3.

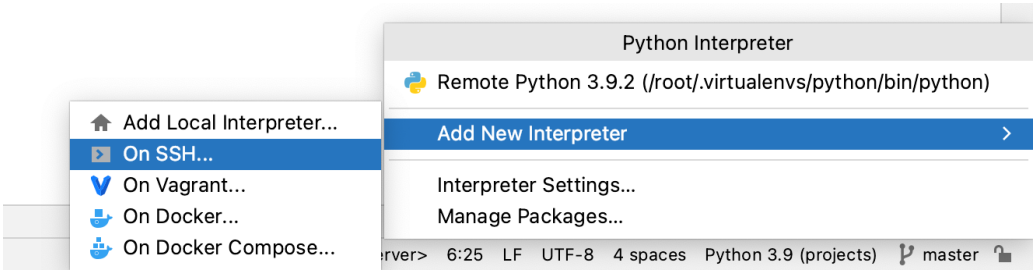

Abbildung 3: Einen neuen Python Interpreter hinzufügen

Im Anschluss können Sie Ihre VM wie in Abbildung 4 angeben und Ihren eigenen SSH Key eintragen (Abbildung 5).

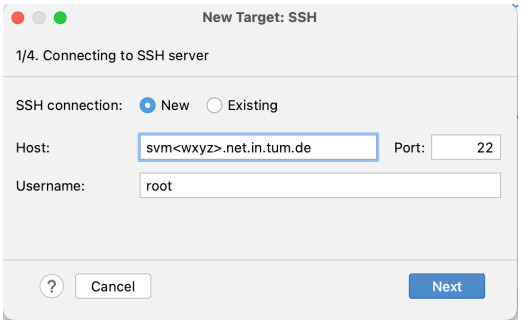

Abbildung 4: Tragen Sie hier Ihre VM ein

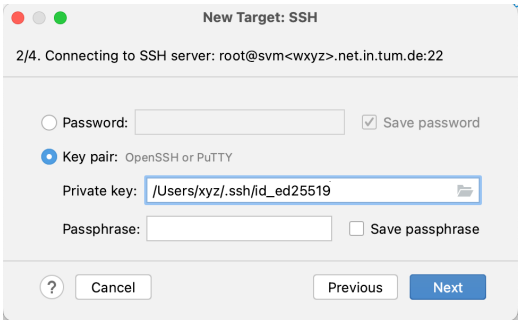

Abbildung 5: Geben Sie hier Ihren SSH key an

Im Anschluss werden Sie gefragt, ob eine virtuelle Python Umgebung initialisieren wollen, wir empfehlen einfach die Standardeinstellungen zu belassen wie in Abbildung 6. Diese Umgebungen helfen mögliche (Versions-)Konflikte zwischen den genutzten Python Paketen in den folgenden Assignments vorzubeugen.

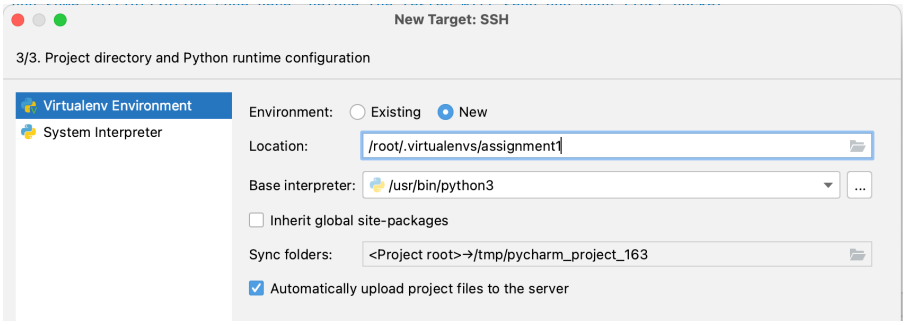

Abbildung 6: Es besteht die Möglichkeit eine virtuelle Python Umgebung für das Projekt einzurichten

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, *Installieren von Linux unter Windows mit WSL*. https://learn.microsoft.com/de-de/windows/wsl/install.

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

[7] Microsoft, "Remote Development using SSH," 2023. https://code.visualstudio.com/docs/remote/ssh.

[8] JetBrains, "Configure an interpreter using SSH." https://www.jetbrains.com/help/pycharm/configuring-remote-interpreters-via-ssh.html.