OS-UE

Praktische Übung 1

Georg Merzdovnik Aljosha Judmayer Gabriel Gegenhuber
Michael Pucher

1 Lab Beschreibung

Diese Übung wurde ursprünglich von Georg Merzdovnik (SBA Research) und Aljosha Judmayer (SBA Research/Uni Wien) entwickelt und von uns für Betriebssysteme UE erweitert. Bei Fragen bitte das Moodle-Forum benutzen. Falls es Fragen zur Abgabe selbst gibt, bitte Michael Pucher oder Gabriel Gegenhuber kontaktieren (die anderen Übungsleiter sind nicht in die praktische Aufgaben involviert), E-Mail Adressen:

- m.pucher@univie.ac.at
- gabriel.karl.gegenhuber@univie.ac.at

Bitte allerdings nur Fragen per E-Mail stellen, wenn die Frage einen konkreten Lösungsansatz enthält, der nicht in Moodle aufscheinen soll, oder es um individuelle Probleme der Abgabe geht (beispielsweise, warum die abgegebenen Log-Files 0 Punkte ergeben). Generell gilt, wenn ein abgegebenes Log-File 0 Punkte ergibt, dann ist das ein Fall für eine E-Mail; wenn es um fehlende Punkte bei den Text-Fragen geht, wäre Moodle angebracht.

Alle Übungen sind auf *Debian GNU/Linux 11.3 amd64* ausgelegt. Im Zuge des ersten Übungsabschnittes (`setup`) wird eine entsprechende Virtuelle Maschine aufgesetzt, die für alle weiteren Übungsabschnitte benutzt werden kann.

1.1 Manpages

Um auf Unix/Linux-Systemen Hilfe zu einem Systemprogramm und den verfügbaren Parametern zu erhalten kann man sich über die Kommandozeile mittels `man(1)` die Programmdokumentation (manual page) anzeigen lassen.

```
$ man man    # zeigt die manpage zu man :)
$ man ls     # zeigt die manpage zu ls (list directory content)
```

Auch von Programmen die manuell über die Paketverwaltung installiert werden gibt es in der Regel gut gepflegte Dokumentation, die bei der Installation automatisch mitinstalliert und mittels `man(1)` abgefragt werden kann.

Während die Anzeige via `man(1)` auch auf Systemen die offline sind verfügbar ist, sind Manpages auch im Internet einsehbar und sind eine beliebte Informationsquelle um sich über die Funktionen und Parameter eines Programms zu informieren. Eine Hilfreiche Website um Hilfe zu konkreten oder komplexen Programmaufrufen zu erhalten ist explainshell.com. Nach Eingabe eines konkreten Programmaufrufs werden die relevanten Teile der Manpage angezeigt (z.B. hier für das entpacken eines Archivs).

1.2 Umgebungsvariablen

Umgebungsvariablen sind konfigurierbare Variablen, die von einem oder mehreren Prozessen verwendet werden können um z.B. das Programmverhalten zu beeinflussen. Windows-User kennen

das Prinzip z.B. vom setzen der PATH-Variable (damit neu installierte Programme auch global bekannt und verfügbar sind).

Während Umgebungsvariablen in der Regel global verfügbar und für langfristigen Einsatz gedacht sind, gibt es auch noch kurzlebige Shell-Variablen, die nur in der aktuellen Shell bzw. im aktuellen Skript verfügbar sind.

Im Zuge von Skripts oder Programmaufrufen kann mittels `$`-Zeichen (= Auswertung) auf die Variable zugegriffen werden.

```
$ OSUE=1337          # setzen der variable
$ echo $OSUE         # ausgabe des variableninhalts
$ touch $OSUE.txt    # zugriff auf die variable mittels programmaufruf
```

1.3 NAME und MATR

- Wenn in der Übungsbeschreibung `$NAME` verwendet wird, soll an dieser Stelle der u:account Username verwendet werden. Dieser enthält keine speziellen Zeichen und entspricht daher automatisch allen nötigen Bedingungen für einen Username unter Debian. Weitere Infos zum Thema Username-Konventionen sind unter `man adduser` oder hier zu finden.
- Wenn `$MATR` in der Übungsbeschreibung verwendet wird, soll an dieser Stelle die eigene Matrikelnummer eingesetzt werden, z.B. 'MATR="123456789".

1.4 LAB und TASK

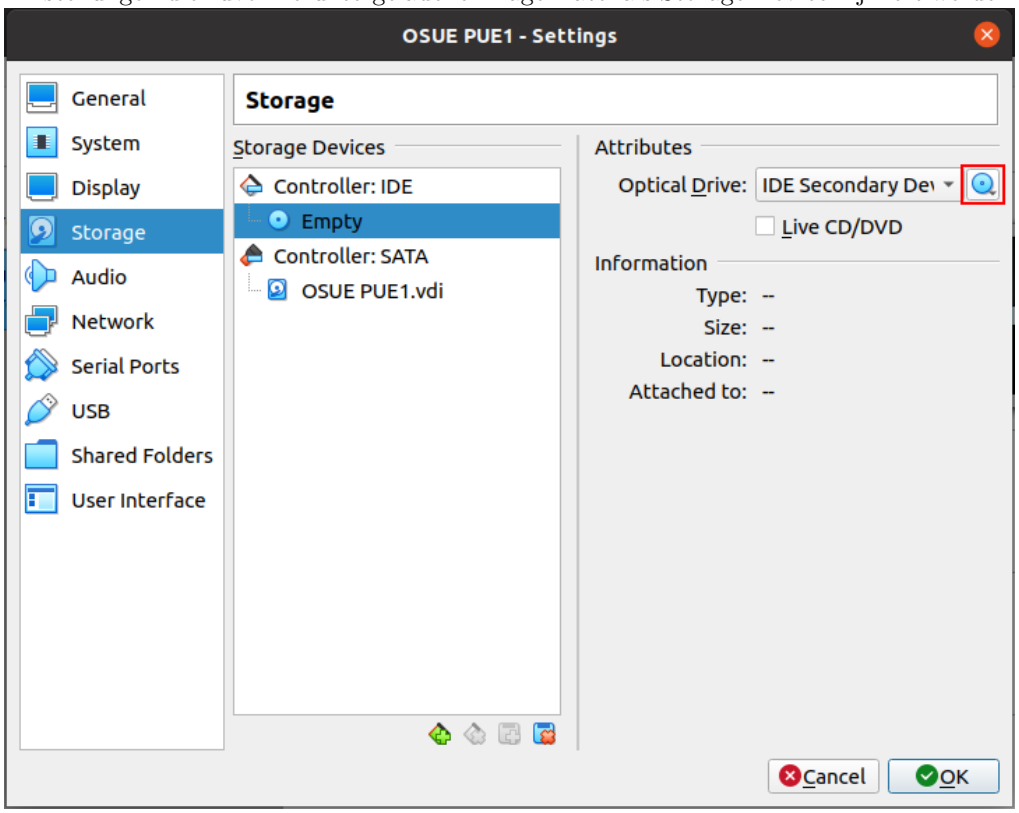
- Wenn `$LAB` in der Übungsbeschreibung verwendet wird, bezieht es sich auf den Namen des aktuellen Übungsabschnittes. Beispiel: Wenn der Übungsabschnitt "Setup" heißt, dann gilt `LAB="setup"`. Der genaue Wert für diese Variable wird im entsprechenden Übungsabschnitt genannt.
- Ein Übungsabschnitt kann aus mehreren Aufgaben bestehen, wo jeder Unterabschnitt einer Aufgabe entspricht. Wenn `$TASK` in der Beschreibung verwendet wird, bezieht es sich auf den Namen des Unterabschnitts. Der Name der Aufgabe wird nochmals genauer genannt. Beispiel: `TASK="install"`.

2 Setup

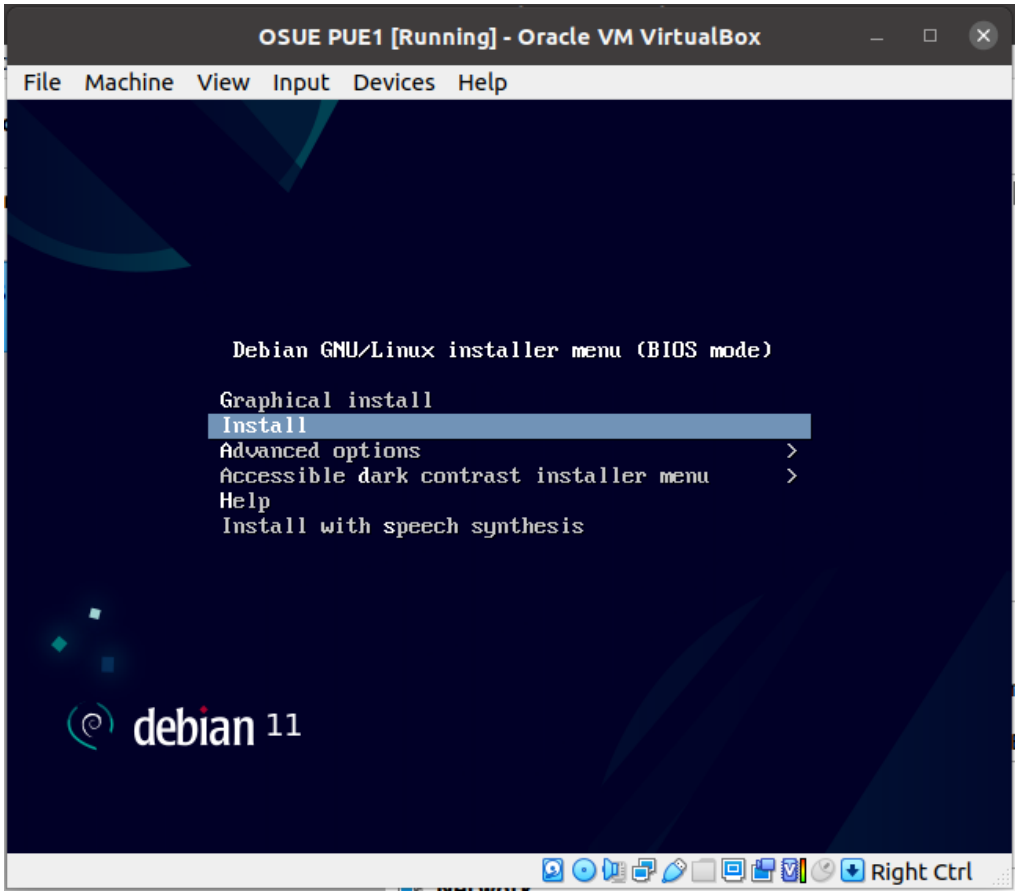
- `LAB=setup`

2.1 Linux Installation

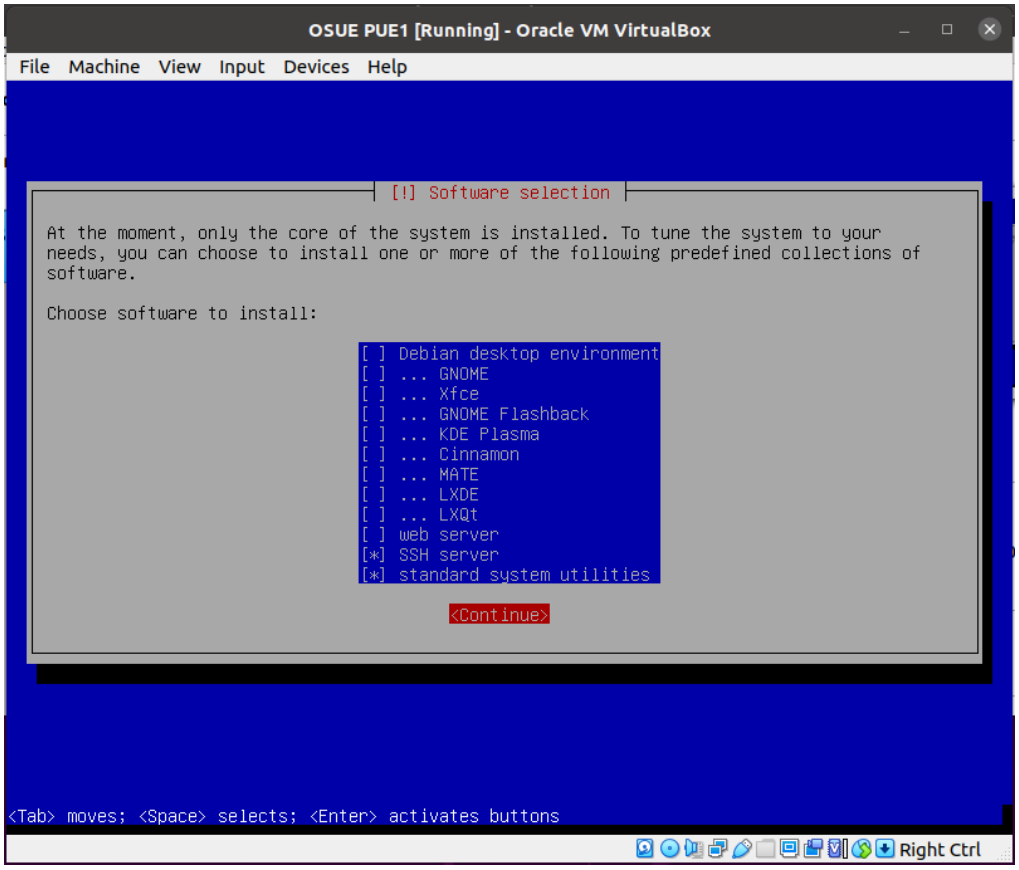
1. Starte dein Betriebssystem und installiere VirtualBox. Grundsätzlich darf für diese Übung auch andere Software zur Virtualisierung (z.B. VMware oder QEMU) verwendet werden, wir können im Forum aber keinen Support dafür garantieren.
2. Lade von der Debian-Website das aktuelle *Debian GNU/Linux 11.3 amd64* netinstall *.iso* Image herunter
3. Verifiziere die SHA256 Prüfsumme deines Downloads um die Authentizität und Integrität deines Downloads sicherzustellen.
   - Auf Linux-basierten Systemen (oder in der WSL Bash) kann `sha256sum` dafür verwendet werden. Analog dazu gibt es auf Windows (Powershell) `Get-FileHash` oder `shasum -a 256` auf macOS.
   - Nähere Infos dazu gibt es unter https://www.debian.org/CD/verify.en.html
4. Setze eine neue Virtuelle Maschine über VirtualBox auf und installiere *Debian GNU/Linux 11.3 amd64*.
   - Dazu muss eine neue VM (mit Standardeinstellungen) angelegt werden und danach in den Einstellungen die zuvor heruntergeladene Image Datei als Storage Device injiziert werden.



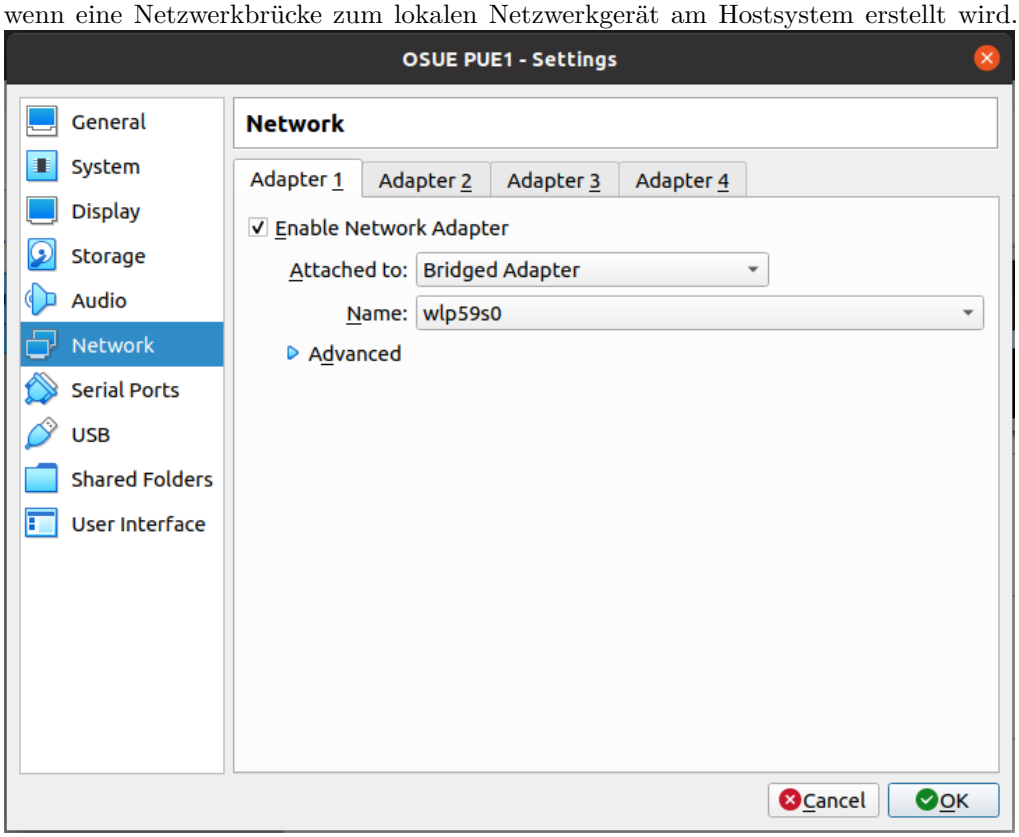
   - Nachdem die VM mit dem Installations-Image gestartet wurde wählen wir das Kommandozeilen-basierte Setup um eine authentische Linux-Experience zu garantieren. :)



- Während des Setups genügt es die meisten Einstellungen auf den vordefinierten Standardsettings zu belassen. Alle notwendigen Änderungen sind wie folgt zu treffen:
- Der Hostname des Systems muss auf `$MATR` gesetzt sein
- Im Zuge des Setups soll ein User mit den Usernamen `$NAME` angelegt werden
- Sowohl für den *root* User, als auch für den persönlichen User soll ein sicheres Passwort verwendet werden
- Die VM soll ohne Grafische Oberfläche, und mit aktiviertem SSH Server installiert werden.



- Damit die VM via `ssh`/`scp` vom Host-System aus zugänglich ist, müssen die Netzwerkeinstellungen in den VM Settings geändert werden. Am einfachsten ist dies zu erreichen, wenn eine Netzwerkbrücke zum lokalen Netzwerkgerät am Hostsystem erstellt wird.



Nach einem `reboot` sollte die VM eine eigene IP-Adresse im internen Netzwerk bekommen und von dort aus via `ssh` erreichbar sein. Die zugewiesene IP-Adresse kann auf der VM mittels `ip addr` abgefragt werden.

5. Die VM ist nun einsatzbereit und kann für allen zukünftigen Übungen verwendet werden.

2.1.1 Zugriff auf die Virtuelle Maschine

- Der Zugriff auf die Kommandozeile kann entweder direkt über das VirtualBox Fenster oder via `ssh` erfolgen. Sofern als Hostsystem Windows verwendet wird und kein WSL aktiviert ist muss für den Zugriff via `ssh` noch ein Clientprogramm (z.B. PuTTY) installiert werden.
- Im Zuge der Übungen wird es nötig sein verschiedene Dateien zwischen Hostsystem und Gast-VM hin und her zu kopieren. Eine einfache Lösung dafür ist die verwendung des Systemprogramms `scp(1)`.
- Für das Verwenden von `scp(1)` ist es erforderlich, dass am Zielsystem ein `ssh`-Server läuft und das Zielsystem übers Netzwerk erreichbar ist. Das folgende Beispielkommando kopiert den Ordner `/home/$NAME/lab1` von der Debian-VM in das lokale Arbeitsverzeichnis am Hostsystem wenn es auf diesem ausgeführt wird. `$IP` steht dabei für die IP-Adresse des Debian-Systems.

  ```
  $ scp -r $NAME@$IP:~/lab1 ./
  ```

- Alternativ kann für den Dateitransfer auch ein grafischer Client (z.B. FileZilla) verwendet werden.

2.1.2 script

Die meisten Übungen erfordern die Benützung von `script(1)` um die ausgeführten Befehle zu dokumentieren. Falls nicht anders angegeben, sollte `script(1)` wie folgt ausgeführt werden:

```
$ script --timing="./$MATR.$LAB.$TASK.tm" -fa "./$MATR.$LAB.$TASK.log"
```

Zur Benützung von `script(1)` sollten folgende Dinge im Hinterkopf behalten werden:

- Wenn die Übungsbeschreibung "**script** starten" enthält, sollte `script(1)` wie oben beschrieben ausgeführt werden.
- Ab "**script** starten." oder "Starte script" sollten alle Befehle in der gleichen interaktiven Sitzung ausgeführt werden, um alle Kommandos entsprechend aufzuzeichnen.
- Es darf **niemals** `cat $MATR.$LAB.$TASK.log` in dieser Sitzung ausgeführt werden!!
- `script(1)` sollte beendet werden, wenn in der Übungsbeschreibung der Punkt "**script** anhalten." oder "Stoppe script" erreicht wird. Das kann mithilfe des `exit` Kommandos erreicht werden, wenn es innerhalb einer `script(1)` Sitzung ausgeführt wird.
- Falls Kommandos vergessen oder nicht aufgezeichnet wurden, können diese mit `script(1)` an das bestehende Log-File angehängt werden mit `-a`.
- Der Upload von Log-Files ist größenlimitiert auf 10MB, enthält ein Log-File größere unerwartete Daten, sollten diese vor dem Upload entfernt werden.

Für weitere Informationen siehe `man script`.

2.1.3 start.sh

- Statt `script(1)` manuell zu verwenden, stellen wir auch das Helfer-Skript `start.sh` bereit. Dieses Skript fragt nach dem Namen des aktuellen `$LAB` und `$TASK`, und startet dann automatisch eine interaktive `script(1)` Sitzung. Die entsprechenden Log-Dateien werden in das aktuelle Arbeitsverzeichnis ausgeschrieben.

- `start.sh` setzt ausserdem bestimmte Einstellungen in `.bashrc` und `.vimrc` für den aktuellen User. Das könnte im Verlauf der Übung hilfreich sein.

2.1.4 Abgabe

Nach dem Abschluss eines Übungsabschnitts muss die Lösung an unseren Abgabeserver geschickt werden. Dazu stellen wir Ein Python-Skript (`submit.py`) zur Verfügung.

Beim erstmaligen Ausführen des Abgabeskripts muss der persönliche Abgabetoken (siehe Feedback bei der Abgabe in Moodle) gesetzt werden.

```
$ ./submit.py reset_token
```

Neben der Abgabe der Logs und etwaiger Dateien, die im Zuge des Übungsabschnitts erzeugt wurden gibt es auch Fragen die zu jedem Übungsabschnitt beantwortet werden müssen.

Das Abgabescript erstellt die dafür vorhergesehenen Antwortdateien (`$MATR.$LAB.txt`) mittels

```
$ ./submit.py fetch_questions
```

Nach dem Abschluss eines Übungsabschnitts und dem Einfügen der Antworten in die dementsprechende Antwortdatei muss das Abgabeskript mit folgendem Befehl aufgerufen werden

```
$ ./submit.py submit
```

Das Skript überprüft die abgegebenen Lösungen und zeigt die aktuelle Punkteübersicht an. Insgesamt können für die ganze Übung 15 Punkte (3 Punkte pro Übungsabschnitt) erreicht werden. Die Abgabe einer falschen Lösung hat keine negativen Auswirkungen. Bei mehrmaligen Abgaben zählt die Abgabe in der die meisten Punkte erreicht wurden.

2.2 Paketverwaltung

- `TASK=pkgmgt`

1. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
2. **Starte script**
3. Installiere alle Updates die für dein Debian-System verfügbar sind (verwende `su` um root-Rechte zu erlangen).
4. Verwende den Paketmanager (`apt`) um `sudo` zu installieren (verwende `su` um root-Rechte zu erlangen):
5. Verwende den Paketmanager (`apt`) um folgende Software zu installieren (verwende nun `sudo` anstatt `su`):
   - `vim`
   - `apt-file`
6. **Stoppe script**

Hilfreiche Kommandos: `apt`, `apt-get`, `apt-cache`, `apt-file`, `dpkg`, `uname`, `lsb_release`, ...

2.2.1 Fragen (Abgabeskript)

- Welche beiden (apt) Kommandos hast du verwendet um alle Systemupdates einzuspielen?
- Welche vim Version wurde über die Paketverwaltung installiert?
- Im Zuge der Installation von vim wurden zwei weitere Pakete als Abhängigkeiten installiert. Welche waren es?
- Welche Linux Kernel Version läuft aktuell auf dem System?
- Welche Debian-Version ist am System installiert und was ist der Versions-Codename?

2.3 Kommandozeilen-Editor

- `TASK=edit`

Beim Arbeiten auf Linux-Systemen hat man (wie auch in dieser Übung) oft nur die Kommandozeile (CLI) zur Verfügung. Bei der viel diskutierten Frage "Was ist der beste Editor?" scheiden sich die Geister, weshalb wir diese Entscheidung euch selbst überlassen. In dieser Übung sollt ihr euch deshalb über vorherrschende Kommandozeilen-Editoren auf Linux informieren und den am besten zu euch passenden Editor finden.

1. Wähle einen Kommandozeilen-Editor der zu dir passt. In dieser Übung steht `$EDITOR` für den Namen dieses Editors.
   - pico/nano (anfängerfreundlich)
   - vim (unser Favorit)
   - emacs (mächtig aber schwierig zu lernen)
   - ...
2. **Starte script** und erledige folgende Punkte mit deinem Lieblingseditor:
   1. Erstelle eine neue Datei mit dem Namen `I_love_$EDITOR.txt`
   2. Schreibe einen Grund warum du dich für diesen Editor entschieden hast in die Datei.
   3. Schließe den Editor
   4. Öffne die Datei abermals und bearbeite den Dateiinhalt.
   - Schreibe eine gute und eine schlechte Eigenschaft deines Lieblingseditors in die Datei.
   - Lösche die schlechte Eigenschaft wieder aus der Datei.
   5. Speichere die durchgeführten Änderungen und schließe die Datei.
   6. **Stoppe script**

Hilfreiche Kommandos: `apt-get`, `vim`, `nano`, `pico`, `emacs`, ...

2.3.1 Fragen (Abgabeskript)

- Welchen Kommandozeilen-Editor hast du dir ausgesucht?

3 Rechtemanagement

- `LAB=permissions`

In dieser Übung erfährt ihr mehr über das Datei-, Benutzer-, und Rechtemanagement auf Unix/Linux-Systemen.

3.1 Benutzerverwaltung

- `TASK=usermgt`

0. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
1. **Starte script** und führe die folgenden Aktionen in der gleichen Session durch.
2. Wechsle mithilfe `su` zum `root` User und installiere `sudo(1)` falls es noch nicht am System installiert ist.

```
$ su
$ apt-get install sudo
```

3. Füge deinen User `$NAME` zur *sudo* Gruppe hinzu. Du kannst nun `sudo(1)` von deinem normalen User aus verwenden und musst für privilegierte Aktionen nicht mehr extra zum `root`-Benutzer wechseln.
4. Füge zwei neue Benutzerkonten zum System hinzu. Verwende für die neuen User die Benutzernamen *alice* und *bob*.
   - Setze für beide User das gleiche Passwort.
5. Verifiziere dass sich die beiden neuen User einloggen können.

```
# von deinem normalen Benutzerkonto (nicht root) aus
$ su alice  # erfordert die Eingabe des Passworts des neuen Users
$ exit
$ su bob    # erfordert die Eingabe des Passworts des neuen Users
```

6. Schau dir die Dateien `/etc/passwd` und `/etc/shadow` an und vergleiche die entsprechenden Einträge für die beiden neuen User
   - Überlege warum die Eintrage (in `/etc/shadow`) nicht ident sind obwohl beide User das gleiche Passwort verwenden
7. Erstelle eine neue Benutzergruppe mit dem Namen *usershare*.
8. Füge beide Useraccounts (*alice*,*bob*) zu der soeben erstellten Benutzergruppe (*usershare*) hinzu.
9. Logge dich als User *alice* mittels `su alice` ein.
10. Versuche eine Kommando mittels `sudo` auszuführen (z.B. `sudo cat /etc/shadow`).
11. **Stoppe script**

Hilfreiche Kommandos: `adduser(8)`, `addgroup(8)`, `su(1)`, `sudo(1)`, ...

3.1.1 Fragen (Abgabeskript)

- Welcher Eintrag wurde in `/var/log/auth.log` geschrieben, nachdem *alice* auf `/etc/shadow` zugegriffen hat?

3.2 Dateirechte

- `TASK=fileperm`

0. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
1. **Starte script**
2. Log dich mittels `su alice` als User *alice* ein.
3. Erstelle im Home-Verzeichnis eine Datei (z.B. mittels `touch(1)` oder mit Hilfe deines Lieblingseditor).

4. Verändere die Zugriffsrechte für die Datei, sodass:
   - Nur das Benutzerkonto, das die Datei besitzt, Lese-, sowie Schreibrechte hat
   - Alle Mitglieder der Gruppe *usershare* dazu berechtigt sind die Datei zu lesen
   - Alle anderen Konten keine Rechte besitzen
5. Nachdem die Berechtigungen richtig gesetzt wurden, führe folgendes Kommando (inklusive Kommentar `# done`) aus, damit die Zeile einfach im Logfile gefunden werden kann.

```
$ ls -l $YOURFILENAME # done
```

6. Versuche nun die Datei von den Benutzerkonten *alice*, *bob*, von deinem persönlichen Benutzerkonto `$NAME` und vom *root*-Benutzer aus zu lesen und schreiben. (verwende `su` zum Wechseln zwischen den Benutzerkonten).
7. **Stoppe script**

Hilfreiche Kommandos: `chown(1)`, `chmod(1)`, `su(1)`, `touch(1)`, ...

3.3 Ordnerrechte

- `TASK=dirperm`

0. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
1. **Starte script**
2. Log dich mittels `su alice` als User *alice* ein.
3. Erstelle im Home-Verzeichnis mittels `mkdir(1)` einen neuen Ordner.
4. Verändere die Zugriffsrechte für den Ordner, sodass:
   - Alle Mitglieder der Gruppe *usershare* Dateien lesen und schreiben dürfen
   - Dateien nur von ihren Besitzern gelöscht werden dürfen (analog zum `/tmp` Verzeichnis)
   - Dateien in diesem Ordner automatisch zur Gruppe *usershare* gehören.
   - Andere User nicht in der Lage sind Dateien im Ordner zu lesen/schreiben oder ins Verzeichnis zu wechseln.
5. Nachdem die Berechtigungen richtig gesetzt wurden, führe folgendes Kommando (inklusive Kommentar `# done`) aus, damit die Zeile einfach im Logfile gefunden werden kann.

```
$ ls -ld $YOURDIRNAME # done
```

6. Versuche nun in dem Ordner von den Benutzerkonten *alice*, *bob*, von deinem persönlichen Benutzerkonto `$NAME` und vom *root*-Benutzer aus zu lesen und schreiben. (verwende `su` zum Wechseln zwischen den Benutzerkonten).
7. **Stoppe script**

Hilfreiche Kommandos: `chown(1)`, `chmod(1)`, `su(1)`, `touch(1)`, `mkdir(1)`, ...

4 Dateisystem

- `LAB=filesystem`

In dieser Übung geht es über die Konfiguration von Dateisystemen unter Linux. Eine Festplatte kann in mehrere *Partitionen* geteilt werden. Jede Partition bildet eine logische Einheit, die separat von anderen Partitionen auf der gleichen Platte verwendet werden kann. Damit eine Partition vom Betriebssystem aktiv genutzt werden kann, muss ein entsprechendes *Dateisystem* definiert werden. Das Dateisystem gibt vor, in welchem Format die Daten auf der Partition abgelegt werden.

Mehr Informationen zu Filesystemen und Partitionierung gibt es unter anderem hier:

- https://wiki.archlinux.org/index.php/Partitioning
- https://wiki.archlinux.org/index.php/File_Systems
- https://wiki.debian.org/FileSystem
- https://wiki.debian.org/FilesystemHierarchyStandard

4.1 Arbeiten mit Dateisystemen

- `TASK=fs`

1. Füge eine neue Festplatte zu deiner *Debian GNU/Linux 11.3 amd64* VM hinzu. Diese neue Festplatte soll `128` MByte groß sein.
   - Das Hinzufügen neuer Festplatten ist über die Einstellungen deiner VM (in VirtualBox) möglich.
2. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
3. **Starte script**
4. Erstelle eine Partition auf der neu hinzugefügten Festplatte.
5. Formatiere die soeben erstellte Partition mit einem beliebigen Dateisystem (z.B. ext4).
6. Binde das neue Dateisystem in dein System ein (mount).
7. Erstelle ein paar Dateien auf dem soeben eingebundenen Dateisystem.
8. Stelle sicher, dass das neue Dateisystem beim Booten des Systems automatisch eingebunden wird. Zum Testen darf das **script** natürlich via `exit` angehalten werden. Nach dem Reboot muss es jedoch wieder ausgeführt werden (**Starte script**), damit die Aufzeichnung im Logfile wieder aufgenommen wird.
9. Hänge das Dateisystem wieder aus (unmount).
10. Erstelle ein ein vollständiges Abbild der Partition.
11. Lösche den Inhalt der Partition indem du sie mit Nullen überschreibst.
12. Stelle sicher, dass die Partition überschrieben wurde, indem du einen Dump der ersten 512 Bytes erstellst und diesen mit einem Hex-Editor überprüfst.
13. Binde das zuvor erstellte Abbild mit `losetup(8)` ins System ein und kopiere eine Datei in das Abbild.
14. Stelle sicher, dass die Änderungen wirksam und das Abbild neu gespeichert wurde.
15. **Stoppe script**
16. Führe einen System-Reboot aus und überprüfe ob alles okay ist.

Hilfreiche Kommandos: `fdisk(8)`, `mkfs(8)`, `mount(8)`, `dd(1)`, `xxd(1)`, `wipe(1)`, `shred(1)`, `losetup(8)`, `fstab(5)`, ...

4.1.1 Fragen (Abgabescript)

- Zum Überschreiben deiner Partition mit Nullen hast du vermutlich `/dev/zero` verwendet. Welche drei ähnlichen Device-Files gibt es analog zu `/dev/zero` noch?'

5 Pipes & Filter

- `LAB=pipesfilters`

In dieser Übung geht es darum das Verwenden von Pipes `|` und verschiedener Filter (z.B. `wc(1)`,`sed(1)`,`awk(1)`,`head(1)`,`tail(1)`...) zu erlernen.

5.1 Log Analyse 1

- `TASK=loganalysis_1`

0. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
1. **Starte script**
2. Verschiebe die Datei `lab_pipefilters.zip` auf deine VM und entpacke den Inhalt der gepackten Datei.
3. Entpacke alle (nicht leeren) *access.log* Logfiles mithilfe einer einzigen Kommandozeile.
   - Das Verwenden von Pipes `|` könnte hilfreich sein
   - Das Verwenden von `xargs(1)` könnte hilfreich sein
   - Das Verwenden von `find(1)` könnte hilfreich sein
   - ...
4. Gib eine Liste mit der jeweiligen erste Zeile aus allen Logfiles aus (`head(1)`).
5. Gib eine Liste mit der jeweiligen letzten Zeile aus allen Logfiles aus (`tail(1)`).
6. Jeder Browser identifiziert sich mittels *User-Agent*-String, der am Ende einer Zeile im Logfile steht, z.B.:

```
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)"
```

Gib eine Liste aller in den Logifiles vorkommenden *User-Agent*-Strings aus. Wenn ein *User-Agent* mehrmals vorkommt, soll er nur ein einziges Mal ausgegeben werden. + Es ist erlaubt die Transformationen auf mehrere Zeilen aufzuteilen, obwohl es auch mit einem Einzeiler machbar ist ;) + Bei der Lösung der Aufgabe könnten die unten aufgelisteten Systemprogramme und das verwenden einer Regular Expression helfen + Speichere das Ergebnis in eine Datei mit dem Namen `ua_uniq.txt` 7. **Stoppe script**

5.1.1 Fragen (Abgabescript)

- Wie viele Logeinträge (= Zeilen) wurden insgesamt in den Logfiles aufgezeichnet?
- Wie viele Bytes hat das größte Logfile?

5.2 Log Analyse 2

- `TASK=loganalysis_2`

0. **Starte script**
1. Erstelle eine neue Datei (`ua_frequ.txt`) die ein Ranking aller *User-Agents* aus der vorherigen Aufgabe beinhaltet. Dabei soll jeder *User-Agent* genau einmal vorkommen und am Beginn jeder Zeile die Anzahl der Vorkommnisse in den Logfiles stehen. Die Liste soll außerdem nach Häufigkeit sortiert sein und somit mit dem *User-Agent*, der am öftesten vorkommt starten, z.B.:

```
13 Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
```

2. Transformiere die Datei `ua_frequ.txt` in eine weitere Datei `ua_frequ.csv` die folgendermaßen formatiert ist:

```
nr,ua
13,"Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)"
```

Wie vorhin soll die resultierende Liste auch wieder nach absteigender Häufigkeit sortiert sein.

3. Finde heraus wie viele verschiedenen IP Adressen den *User-Agent* `Python-urllib/2.7` verwenden.
4. Erstelle eine Datei für jede Quell-IP-Adresse (`$IP.log`), die irgendwann den *User-Agent* `Python-urllib/2.7` verwendet hat. Die resultierenden Dateien sollen alle Anfragen von der jeweiligen IP-Adresse beinhalten (unabhängig vom *User-Agent*, der in den anderen Anfragen der jeweiligen IP verwendet wurde).
5. Erstelle eine neue Datei (`method.log`), die alle Anfragen, die **nicht** GET-, POST- oder HEAD-Anfragen sind beinhaltet. In dieser Datei soll jede Anfragezeile (ohne IP, Timestamp oder User-Agent) nur ein einziges mal angezeigt werden.
    - Erstelle wieder eine neue Liste, die alle vorkommenden Anfragezeilen mit einem Zähler (Anzahl der Vorkommnisse) am Zeilenbeginn beinhaltet.
6. **Stoppe script**

Hilfreiche Kommandos: `unzip(1)`, `gzip(1)`, `find(1)`, `xargs(1)`, `grep(1)`, `sort(1)`, `uniq(1)`, `wc(1)`, `sed(1)?`, `awk(1)`, ...

5.2.1 Fragen (Abgabeskript)

- Welche IPs verwenden den User-Agent `Python-urllib/2.7`?

6 Prozessmanagement

- `LAB=processmgt`

In dieser Übung geht es um Prozessmanagement und den Prozesslebenszyklus unter Linux.

6.1 Zombieland

- `TASK=zombieland`

0. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein.
1. Verschiebe die Datei `lab_processmangement.tar.gz` auf deine VM und entpacke den Inhalt der gepackten Datei.
2. **Starte script**
3. Starte das Programm `zombieland` im Hintergrund mit Hilfe von `&`:

```
$ ./zombieland 1000 &> ./output &
```

4. Verwende nun das `ps` Kommando um nach allen `zombieland` Prozessen zu suchen und die folgenden Informationen zu sammeln.
   - Prozess-ID (pid)
   - Prozess-ID des Elternprozesses (ppid)
   - Aktueller Prozessstatus
   - Den Kommandozeilenparameter die verwendet wurden um den Prozess auszuführen
   - Der *resident set size* des Prozesses
5. Verwende das `kill` Kommando und versuche den *zombie*-Prozess (Status `Z`) mithilfe folgender Signale zu terminieren:
   - `SIGTERM`
   - `SIGKILL`
6. Verwende das `kill` Kommando und sende ein `SIGCHLD` Signal zum Elternprozes des *zombie*-Prozesses.
   - Was wird vom `zombieland` Programm ausgegeben?
7. Wiederhole die Schritte **3.** and **4.**. Starte das Programm `zombieland` erneut und suche nach allen `zombieland`-Prozessen um die erforderlichen (p)pids zu bekommen.
8. Verwende das `kill` Kommando um ein `SIGTERM` Signal zum Elternprozes des *zombie*-Prozesses zu senden.
   - Was wird vom `zombieland` Programm ausgegeben?
   - Warum ist die Ausgabe anders als vorhin?
9. Wiederhole Schritt **3.** einige Male und starte mehrere Instanzen des `zombieland` Programms.
10. Sende das `SIGKILL` Signal zu allen Instanzen mithilfe des `killall` Kommandos.
11. **Stoppe script**

Hilfreiche Kommandos: ps, pgrep, kill, pkill, killall, top, htop

6.1.1 Fragen (Abgabeskript)

- Was wurde vom zombieland Programm ausgegeben nachdem es ein SIGCHLD Signal (siehe Punkt 6) bekommen hat?
- Was wurde vom zombieland Programm ausgegeben nachdem es ein SIGTERM Signal (siehe Punkt 8) bekommen hat?

6.2 htop

- `TASK=htop`

0. Boote in deine *Debian GNU/Linux 11.3 amd64* VM und logge dich mit deinem persönlichen Benutzerkonto ein. **Es muss kein script gestartet/gestoppt werden.**

1. Verwende das Kommando `htop` um Informationen über die laufenden Prozesse zu erhalten. Wechsel mithilfe der interaktiven Hilfsfunktion *h* oder *?* in eine Baumansicht und lasse den basename der Prozesse hervorheben.
2. Betrachte den `Nice`-Wert der laufenden Prozesse. Navigiere zum aktuellen `htop` Prozess und erhöhe den `Nice`-Wert des Prozesses auf 10.
3. Navigiere zum aktuellen `htop` Prozess und beende ihn über F9 mit einem `SIGTERM` Signal.

Hilfreiche Kommandos: bash(1), jobs, bg, fg, ps, disown, shopt, nohup(1), pgrep(1), kill(1), pkill(1), top(1), htop(1)

6.2.1 Fragen (Abgabescript)

- Welcher `Nice`-Wert wird standardmäßig für einen neuen Prozess gesetzt?