Studienprogramm Sachbearbeiter:in Digitale Forensik
Praktikum Digitale Forensik Grundlagen
Dozenten: Tim Wetterau | Leander Hossfeld
wetterau@hs-mittweida.de | hossfeld@hs-mittweida.de
Stand: 25.03.2024

HOCHSCHULE MITTWEIDA
University of Applied Sciences

Digitale Forensik

Teil 2 – Grundlagen Linux II

In diesem Praktikum lernen Sie einige weitere Grundlagen zur Verwendung von Linux insbesondere bei der reinen Verwendung in der Kommandozeile. Hierzu nutzen wir wieder die bereits eingerichtete virtuelle Umgebung mit der VM Linux-Mint-BKA. Wir gehen davon aus, dass Sie stets in der Lage sind sich über Optionen der einzelnen Kommandos zu belesen, um die gestellten Aufgaben den Anforderungen entsprechend zu lösen.

Gegenstand dieses Praktikums sind folgende Punkte:

- Einfache Arbeit mit Dateien
- Nutzung von Texteditoren
- Nutzen von Operatoren
- Grundlagen Programmstrukturen

Vorbereitung

Zur Durchführung des Praktikums ist vorausgesetzt, dass die durch die Praktikumsanleiter zur Verfügung gestellte virtuelle Maschine Linux-Mint-BKA oder eine andere funktionierende Linux-Distribution auf ihrem Rechner erfolgreich installiert ist. Des Weiteren sollten Sie die in Praktikum 1 dargestellten Befehle und Operationen verstehen und anwenden können. Natürlich können Sie unterstützend immer das Praktikum 1 mit heranziehen.

Hinweise:

➔ Auch für dieses Praktikum gilt für fortgeschrittene Nutzer, dass Sie dieses Praktikum überspringen können
➔ Wir weisen außerdem darauf hin, dass wir Ihnen abschließend zu den vorgestellten Befehlen eine Befehlsliste mit den wichtigsten zur Verfügung stellen
  - Dennoch sollen Sie dazu angehalten werden stets die Manpages der Kommandos zu konsultieren
  - Die Befehlsliste wird die Möglichkeiten und Mächtigkeit einzelner Befehle nicht vollumfänglich darstellen
  - Arbeiten Sie diese im Zuge des Praktikums im Modul „Betriebssysteme“ weiter aus und vervollständigen Sie diese für sich im Selbststudium.

Nutzen von Operatoren

Eine deutliche Vereinfachung in der täglichen Arbeit bieten Operatoren zur Umleitung von Ausgaben und Steuerung von Befehlsketten. Der einfachste Operator ist der zur Umleitung der Standardausgabe stout mit „**>**“. Damit kann die Ausgabe in eine Datei umgeleitet werden. **Achtung:** Mit der Nutzung von „**>**“ wird der bestehende Inhalt der Datei überschrieben. Soll der umgeleitete Inhalt nur angehangen werden, nutzen wir den Operator „**>>**“. Ein Beispiel dafür ist:

```
$ cat test.txt > new.file       # wenn new.file existiert, wird deren Inhalt überschrieben
$ cat test.txt >> append.file   # Inhalt aus test.txt wird an append.file angehängt
```

Man kann ebenfalls die Fehlerausgabe in eine Datei umleiten. Das erfolgt mit dem Operator „**2>**“ (zweiten Kanal). Somit wird eine Fehlermeldung bei Misserfolg einer Operation in eine Fehlerdatei umgeleitet. *Wenn Sie sich in Ihrem Home-Verzeichnis befinden, probieren Sie den folgenden Befehl aus:*

```
$ cp test.txt folder/test.txt 2> error.log
```

Beschreiben Sie die Funktionsweise des Befehls. Welche Ausgabe erwarten Sie? Überprüfen Sie Ihre Erwartungen im Dateisystem! Warum können Sie das geschehene Verhalten beobachten? Beheben Sie den Fehler, sodass keine Fehlermeldung mehr ausgegeben wird!

Zwei logische Operatoren sind „**&&**“ und „**||**“. Sie stellen das logische **UND** und **ODER** dar. Damit lassen sich mehrere Befehle verknüpfen. Verwenden wir „**&&**“ mit der untenstehenden Syntax, dann wird **Befehl2** nur ausgeführt, wenn **Befehl1** erfolgreich bearbeitet werden konnte. Hingegen ermöglicht „**||**“, dass **Befehl2** nur dann ausgeführt wird, wenn **Befehl1** nicht fehlerfrei (erfolgreich) ausgeführt wurde.

```
$ <Befehl1> && <Befehl2> $
$ <Befehl1> || <Befehl2>
```

Ein weiterer essenzieller Operator ist die Pipe „**|**“. Mit ihr ist es möglich, eine Ausgabe direkt an einen anderen Befehl weiterzuleiten, anstatt ans Terminal. Damit kann **Befehl2** direkt das Ergebnis von **Befehl1** weiterverarbeiten, ohne dass eine Zwischenspeicherung stattfindet. Damit können Sie z.B. einen Befehl schreiben, der alle Namen von Paketen, für die Updates verfügbar sind, in eine Datei schreibt. Dieser könnte wie folgt aussehen:

```
$ sudo apt update | tee upgradable.pkg
```

Einfache Arbeit mit Dateien

Die einfachste Art zur Interaktion mit Dateien ist das Ausgeben des Inhaltes der Datei. Dafür gibt es verschiedene Befehle, die je nach Benutzervorlieben oder Anwendungszweck angewendet werden. Der einfachste ist der Befehl:

```
$ cat [-option] <Dateiname>
```

Damit wird der Inhalt einer Datei in den Standardausgang ausgegeben. Normalen falls ist das die Kommandozeile selbst. Also geben wir schließlich den Inhalt einer Datei auf die Kommandozeile aus. *Geben Sie den Inhalt der Datei test.txt in Ihrem Home-Verzeichnis aus, sodass jede Zeile eine Zeilennummer bekommt und Absatzzeichen sichtbar werden.*

Eine Alternative zur statischen Ausgabe des Testes ist das Ansehen des Inhaltes mit einem interaktiven Viewer. Dazu bietet sich der Pager **more** an. Damit kann man seitenweise den Inhalt ausgeben und von Seite zu Seite blättern. Ist der auszugebene Inhalt größer als eine Bildschirmseite, wird automatisch in den interaktiven Modus gewechselt. Dieser kann mit der Steuerung in **Tabelle 1** bedient werden. Die Syntax wird folgend beschrieben:

```
$ more <Dateiname>
```

Eine Alternative zu **more** ist **less**. Der Aufruf ist der gleiche, wie für **more**. Nur bietet **less** Vorteile im Gegensatz zu **more**: Anzeigen der Datei, obwohl diese noch nicht vollständig in den Speicher geladen ist, einfacheres Navigieren in den Dateien und intuitivere Suchmöglichkeiten. Die Steuerung wird durch **Tabelle 2** dargestellt.

Tabelle 1: Steuerung des Pagers **`more`**

Taste	Aktion
Leertaste	Eine Bildschirmseite weiter
B	Eine Bildschirmseite zurück
F	Zwei Bildschirmseiten weiter
Enter	Eine Zeile weiter
S	Zwei Zeilen weiter
=	Ausgabe der aktuellen Zeilennummer
/	Suchbegriff eingeben, mit Enter bestätigen
v	vim-Editor öffnen (später mehr dazu)

Tabelle 2: Steuerung des Pagers **`less`**

Taste	Aktion
Pfeiltasten	Zeileweise weiter-/zurückblättern
E / Y	wie Pfeiltasten
Bild auf / ab	Seite vor oder zurück
F / B	wie Bild auf / ab
Leertaste	Seite vor
<Zahl> Z / W	angegebene Zahl von Seite vor / zurück
G	Anfang des Dokuments
Shift + G	Ende des Dokumentes
/	Suchbegriff eingeben (kann Regex sein)

Ebenso kann man nur die ersten oder letzten Zeilen einer Datei ausgeben lassen. Dazu bieten sich die Kommandos

```
$ head <Datei>
$ tail <Datei>
```

an. Durch die Option „**`-n`**“ kann darunter auch gesteuert werden, wie viele Zeilen der entsprechenden Datei ausgegeben werden sollen. Das kann dann sinnvoll sein, wenn man nicht genau, was man in eine Datei geschrieben hat und nicht die komplette Datei ausgeben möchte. *Geben Sie die ersten 20 Zeilen der Datei **`names.dict`** in Ihrem Homeverzeichnis aus.*

Ebenso können per Kommandozeile einfache Textoperationen durchgeführt werden. Darunter zählen folgende Kommandos:

```
$ sort <Datei1> [<Datei2> <Datei3> …]        # sortieren des Inhaltes
$ wc <Datei1> [<Datei2> <Datei3>]            # word count (Zählen von Zeilen und Wörtern)
$ tee <Dateiname>                            # Schreiben von Inhalt in eine Datei
```

Nutzen wir das Kommando **`sort`**, so können wir den Inhalt mehrerer Dateien auf einmal sortiert ausgeben. Das lohnt sich, wenn wir mehrere Dateien in einem Zusammenhang haben und nicht jede einzeln ausgeben wollen. Dabei kann nach verschiedenen Kriterien, wie z.B. nach Wörterbuch, unabhängig von Groß-/Kleinschreibung oder nach Dateigröße sortiert werden.

Per **`wc`** kann die Anzahl von Zeilen, Wörter und Zeichen in einer Datei ausgegeben werden. Dabei erscheint eine Ausgabe nach dem folgenden Schema:

```
$ Zeilenanzahl Wörteranzahl Zeichenanzahl Dateiname
```

Wenn die Ausgaben einzeln erfolgen sollen, können auch entsprechende Flags angegeben werden. Damit könnte man z.B. Variablen belegen, welche in einem Skript verarbeitet werden.

Zuletzt der Befehl **`tee`**. **`tee`** nimmt die Eingabe über den Standardeingang der Kommandozeile und schreibt die Eingabe in eine Datei. **Achtung:** Standardmäßig wird damit der Inhalt einer bereits bestehenden Datei überschrieben. Mit „**`-a`**“ kann die Eingabe auch angehangen werden (append). Folgende Syntax gilt (hier mit dem Pipe-Operator als Eingabe):

```
$ head -n 5 test.txt | tee -a test2.txt
```

Im gezeigten Befehl geben wir die ersten 5 Zeilen der Datei test.txt aus und leiten diese per Pipe an die Standardeingabe von **`tee`** weiter. **`tee`** hängt diese wiederum an die Datei **`test2.txt`** an.

Ein weiterer Befehl ist **`paste`** (einfügen). Mit **`paste`** haben wir die Möglichkeit den Inhalt einer Datei zeilenweise an den Inhalt einer anderen Datei anzuhängen. Haben wir also eine Datei mit Namen und eine andere mit Wohnorten können wir diese mit **`paste`** in einer Ausgabe übersichtlicher zusammenfassen. Die Syntax folgt dem Schema:

```
$ paste [-option] <Datei1> <Datei2>
```

Wiederrum kann man mit **`cut`** verschiedene Spalten einer Datei einzeln ausgeben. Dazu müssen diese aber ein einheitliches Trennzeichen haben (**`Leerzeichen, ,, ;, |, -| TAB,`** etc.). Wir nutzen folgende Syntax:

```
$ cut [-d<Delimiterzeichen>] [-f<Spaltenliste>] <Dateiname>
```

Dabei ist zu erwähnen, dass **`TAB`** das Standardtrennzeichen für **`paste`** und für **`cut`** ist.

Bearbeiten Sie dazu folgende Aufgaben:

1. *Schauen Sie sich die Dateien* **`names.dict`** *und* **`schools.dict`** *in den vorgestellten Viewern an.*
    a. *Machen Sie sich mit deren Anwendung vertraut*
2. *Geben Sie die letzten 10 Zeilen der Datei* **`names.dict`** *aus.*
3. *Wie viele Zeilen haben die beiden Dateien jeweils? Nutzen Sie ein Befehl zum Zählen.*
4. *Fügen Sie zeilenweise die Dateien* **`names.dict`** *und* **`schools.dict`** *zusammen.*
    a. *Trennzeichen Tabulator*
    b. *Schreiben der Ausgabe in eine neue Datei* **`concat.dict`**
5. *Sortieren die entstandene Liste nach Namen.*
6. *Geben Sie die Nachnamen und Orte aus* **`concat.dict`** *aus.*
7. ***Zusatz:*** *Geben Sie erneut Nachnamen und Orte aus und sortieren Sie diese nach den Namen.*
    `a.` *Schreiben Sie das Ergebnis in eine neue Datei* **`sortedConcat.dict`**

Nutzung von Texteditoren

Neben dem Schreiben in Dateien per **`tee`** oder Pipes „|“, können Dateien auch direkt editiert werden. Die beiden bekannten Editoren für die Kommandozeile sind **`vi`** und **`nano`**. Deutlich intuitiver und leichter zu bedienen ist **`nano`**. Hingegen gibt es für **`vi`** die Weiterentwicklung **`vim`**, welche deutlich anpassungsfähiger ist. **`vi`** wird eher weniger verwendet und bedarf einer hohen Einarbeitung. An der Stelle sei darauf verwiesen, dass die Arbeit mit **`vi`** zwar möglich ist, aber meist nicht nötig. Sollten Sie dennoch Interesse an dessen Funktionsweise haben verweisen wir auf die Internetquelle der TU Chemnitz verwiesen.

Wir wollen uns hier auf den Editor **`nano`** beschränken. Rufen wir den Editor auf, legt dieser eine neue Datei an, wenn die angegebene Datei nicht existiert. Anderfalls öffnet er nur die angegebene Datei. Rufen Sie den Editor über den folgenden Befehl auf:

```
$ nano new.file
```

Daraufhin öffnet sich der Editor mit einer neuen Datei und wir können fröhlich drauf los editieren. Grundlegend ist die Steuerung intuitiv und sie können sich mit den Pfeiltasten als auch mit den Bildlauftasten im Editor bewegen. Im unteren Teil sehen Steuerungshinweise. Grundlegende Hilfe erhalten Sie außerdem über die Tastenkombination **Strg + G**. *Schreiben Sie ein paar Zeilen in den Editor.*

Entgegen der Nutzung von Windows, können wir die geänderten Daten nicht mit **Strg + S** speichern. Dazu nutzen wir **Strg + O** (write out). Einige weitere Tastenkombinationen finden Sie in **Tabelle 3.**

Tabelle 3: Tastenkombination zur Bedienung von **nano**

Tastenkombi	Funktion
Strg + O	Speichern
Strg + X	Beenden
Strg + W	Text suchen
Alt + A	Anfangsmarkierung setzen (mit Pfeiltasten weiter markieren)
Alt + 6	Kopieren
Strg + K	Ausschneiden
Strg + U	Einfügen
**Strg + **	Ersetzen

Ebenso bietet sich **nano** standardmäßig dazu an, Bash-Skripte zu schreiben. Es bietet ein entsprechendes Highlighting der Syntax. Bash-Skripte werden allgemein immer in der ersten Zeile begonnen mit:

```
$ #!/bin/bash
```

Damit weiß das Betriebssystem, dass diese Datei mit der Bourne-Again Shell (bash) interpretiert werden muss.