

# Grundlagenpraktikum Netz- und Datensicherheit

## *Thema:* **Linux Basics**

Lehrstuhl für Netz- und Datensicherheit  
Ruhr-Universität Bochum

Versuchsdurchführung: Raum ID 2/168



Betreuung: Marcus Niemitz  
Zusammengestellt von: Dominik Noß  
Stand: 25. November 2021  
Version: 5.0

## 1 Hinweise

Vorraussetzung für den Versuch ist es, die Versuchsbeschreibung und die Befehls-Referenz gelesen und verstanden zu haben. Sie müssen alle Kontrollfragen beantworten können. Sollte es Fragen zum Versuch geben, wenden Sie sich an ihren Betreuer.

Zugangsdaten für die Computer lauten **student:12345** und **root:12345**.

Screenshots zur Dokumentation können mit der „Druck“-Taste angefertigt werden. Sofern nicht anders gefordert, sind Dokumentationen in Form von Listings jedoch präferiert.

## 2 Linux und die Shell

Linux an sich ist der von Linus Torvalds entwickelte *Kernel*, also das Herz eines jeden Linux - Betriebssystems. Dieser Kernel ist das unverzichtbare Bindeglied zwischen der Computer-Hardware und den Programmen auf Benutzer-Ebene, jedoch ohne zusätzliche Software recht nutzlos. Erst durch weitere Software, die mit dem Kernel zusammenarbeiten, wird das System für den menschlichen Nutzer benutzbar. Der Einfachheit halber ist mit „Linux“ meist eine Linux-Distribution gemeint, also ein Paket aus Linux-Kernel und einer Auswahl von Programmen. Es gibt viele hunderte Distributionen mit verschiedenen Zwecken wie Pentesting, besondere Kinderfreundlichkeit, geringe Hardware-Anforderungen und der Einsatz als Media-Center. Neben Server-Clouds und Desktop-Varianten mit aufwändig gestalteten Oberflächen kann Linux sogar auf dem Mikroprozessor einer SD-Karten laufen<sup>1</sup>. Eine Möglichkeit, sich einen Überblick zu verschaffen, bietet <http://distrowatch.com>.

Von der Benutzung von Windows ist man gewöhnt, dass viele Funktionen nur per Maus ansteuerbar sind. Ohne zusätzliche Software wie die PowerShell lässt Windows sich ohne Maus praktisch nicht benutzen. Auch in vielen Linux-Distributionen werden grafische Arbeitsumgebungen bereit gestellt, doch die eigentliche Stärke von Linux liegt darunter verborgen: Ein fundamentaler Baustein in Linux ist die *Shell*, auch *Konsole* genannt. Über Text-Eingabe und -Ausgabe besitzt der Benutzer umfassende Gewalt über das System. Es gibt meist bis zu 7 System- bzw. Login-Shells, auf einer davon läuft die grafische Oberfläche. Diese Shell-Instanzen sind besonders dann hilfreich, wenn ein Prozess Amok läuft oder abstürzt und die grafische Oberfläche einfriert. Denn oft ist es noch möglich, per Tastenkombination zu einer System-Shell zu wechseln, um fehlerhafte Prozesse zu beenden und das Problem zu lösen<sup>2</sup>.

In den Anfängen der Datenverarbeitung durch einzelne Benutzer wurden große Rechner geteilt über *virtuelle Terminals* (VT) bedient. Da diese Bedienungsweise für viele Benutzer, insbesondere für Einsteiger, nicht intuitiv und wenig geeignet für das alltägliche Leben ist, bieten viele Linux-Distributionen grafische Oberflächen (Window Manager) wie Gnome, Mate, KDE, XFCE und viele mehr. Sie unterscheiden sich durch die Bedienungsweise, Aussehen und Funktionsumfang. Es lohnt sich deshalb, verschiedene Distributionen bzw. Desktoppakete auszuprobieren.<sup>3</sup> Insbesondere Server und eingebettete Systeme verzichten oft auf eine solche Oberfläche. Aber viele Funktionen, die eine moderne Desktopumgebung ausmachen, benötigen eine grafische Oberfläche. Die Shell wird dann mittels eines *virtuellen Terminal Emulators* (VTE) aufgerufen. Ein Minimum an Funktionalität bieten z.B. die Window Manager Busybox oder i3<sup>4</sup>, welcher einen Fokus auf die Verwendung von VTEs setzt.

Viele grafische Funktionen in Linux kapseln Konsolen-Eingaben. So kann man z.B. die Bildschirmhelligkeit über einen Schieberegler auf dem Desktop anpassen, was tatsächlich das Schreiben eines Zahlen-Wertes in eine Datei zur Folge hat. Oder das Verbinden zu einem WLAN-Netzwerk mittels Mausklicks hat Aufrufe des Programmes *iwconfig* zufolge. Bei manchen Problem-Stellungen ist es sogar nicht möglich, sie mittels Mausklicks zu lösen und man kommt um die Benutzung der Shell nicht umher. Wichtig zu verstehen ist, dass eine Shell nicht nur eine Erweiterung der grafischen Umgebung ist, sondern das Herzstück eines jeden Linux.

---

<sup>1</sup><http://hackaday.com/2013/09/19/advanced-transcend-wifi-sd-hacking-custom-kernels-x-and-firefox/>

<sup>2</sup><http://askubuntu.com/questions/4408/what-should-i-do-when-ubuntu-freezes>

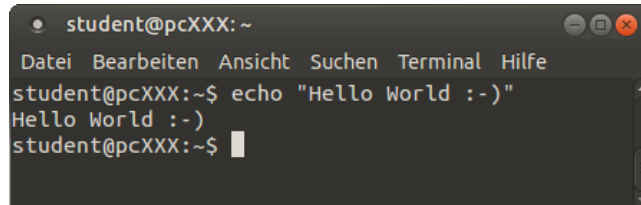
<sup>3</sup><https://www.reddit.com/r/unixporn/>

<sup>4</sup><https://i3wm.org/>

## 3 Arbeiten mit der Shell

### 3.1 Ein- und Ausgabe

Die Shell sieht im Allgemeinen immer gleich aus. Es gibt eine Zeile mit blinkendem Cursor, in welche Befehle getippt werden können. Nach Drücken der Enter-Taste wird der Befehl ausgeführt. Meistens wird die Ausgabe direkt darunter ausgegeben: In manchen Fällen übernimmt der Befehl die Konsole,



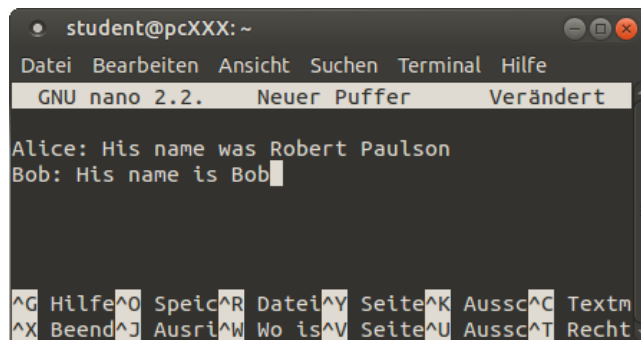
```

student@pcXXX: ~
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
student@pcXXX:~$ echo "Hello World :-)"
Hello World :-)
student@pcXXX:~$

```

wie etwa im Falle von *nano*. Dabei werden Ein- und Ausgabe abgefangen und ein interaktiver Umgang mit dem Programm ermöglicht. Um ein solches Programm zu verlassen, kann bei simplen Programmen mithilfe der Tastenkombination **STRG + C** ein Interrupt-Signal gesendet werden.

*nano* beispielsweise ist ein interaktiver Text-Editor für die Shell. Die ursprünglichste Art des Zugriffs



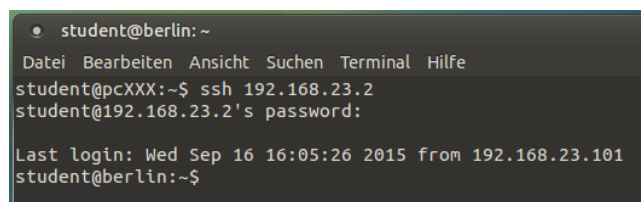
```

student@pcXXX: ~
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
GNU nano 2.2. Neuer Puffer Verändert
Alice: His name was Robert Paulson
Bob: His name is Bob
^G Hilfe^O Speic^R Datei^Y Seite^K Aussc^C Textm
^X Beend^J Ausri^W Wo is^V Seite^U Aussc^T Recht

```

sind die System-Shells. Wenn keine grafische Benutzeroberfläche installiert oder gestartet ist, wird eine solche Shell auf dem Bildschirm angezeigt. In vielen Distributionen kann man System-Shells über Tasten-Kombinationen erreichen. Die erste Shell ist dabei über **Strg+Alt+F1** erreichbar, die zweite über **Strg+Alt+F2** etc. Die sieht - nach dem Login - genauso aus, wie die oben gezeigte Shell, jedoch Bildschirmfüllend und ohne Fenster.

Shells sind auch nicht an die Ein- und Ausgabe-Methoden „Tastatur“ und „Bildschirm“ gebunden. Mittels SSH<sup>5</sup> lässt sich eine Shell auf einem entfernten Rechner starten und kontrollieren. SSH kann weiterhin Public-Key-Cryptography für Authentisierung und Authentifizierung<sup>6</sup> der Benutzer einsetzen und verschlüsselt die übertragenen Daten. Für den Benutzer unterscheidet die SSH-Shell sich optisch nur wenig von einer lokalen: Es gibt viele weitere, vielleicht exotisch anmutende Möglichkeiten, wie



```

student@berlin: ~
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
student@pcXXX:~$ ssh 192.168.23.2
student@192.168.23.2's password:
Last login: Wed Sep 16 16:05:26 2015 from 192.168.23.101
student@berlin:~$

```

etwa der Zugriff über eine RS232-Schnittstelle, über USB oder JTAG oder über Lochkarten.

<sup>5</sup>unter Linux meist OpenSSH: <http://www.openssh.com/>

<sup>6</sup>Der Unterschied: [http://www.korrekturen.de/beliebte\\_fehler/authentifizieren.shtml](http://www.korrekturen.de/beliebte_fehler/authentifizieren.shtml)

## 4 Verschiedene Shells

Es gibt ebenfalls verschiedene Shells, wie *bash*, *zsh*, *ksh* und *ash*. Diese unterscheiden sich in der Funktionalität der Hilfestellungen für den Benutzer. So gibt es verschiedene Auto-Vervollständigungen und Tastenkombinationen. In diesem Versuch wird mit der standardmäßigen „Bourne Again Shell“ *bash*<sup>7</sup> gearbeitet. Für fortgeschrittene Benutzer lohnt es sich oft, einen Blick auf andere Shells zu werfen, da sie eventuell eine angenehmere oder effektivere Arbeitsweise ermöglichen.

## 5 Navigation im Dateisystem

Die Shell befindet sich immer in genau einem Verzeichnis, dem *Arbeitsverzeichnis* (oder *working directory*). Dies ist standardmäßig das Benutzerverzeichnis, welches auch mit `~` referenziert wird. Daher ist im Shell-Prompt die Information zu sehen, dass sich der Benutzer `student` auf dem Gerät mit dem Hostname `berlin` im Verzeichnis `~` befindet. Mittels *pwd* (print working directory) kann der absolute Pfad zum aktuellen Arbeitsverzeichnis ausgegeben werden:

```
1 student@berlin:~$ pwd
2 /home/student
```

Es kann mittels relativer und absoluter Pfadangaben navigiert werden. Bei relativer Pfadangabe bezieht sich der Punkt `.` auf das aktuelle Verzeichnis und `..` auf das darüberliegende. Im folgenden Beispiel wird erst mittels absoluter Angabe in das Verzeichnis `/etc` (beachte: `/` ist das oberste Verzeichnis) gewechselt, dann relativ davon in das Verzeichnis `/etc/ssh` und wieder relativ eine Ebene nach oben.

```
1 student@berlin:~$ cd /etc/
2 student@berlin:/etc$ cd ssh
3 student@berlin:/etc/ssh$ cd ..
4 student@berlin:/etc$ pwd
5 /etc
```

Wenn eine Datei angesprochen werden soll, kann dies ebenfalls relativ oder absolut passieren. Hier wird beispielsweise eine Zeichenkette in die Datei `/tmp/notiz` geschrieben. Danach wird in das Verzeichnis `/tmp` gewechselt und die Datei `notiz` ausgegeben, um den Vorgang zu verifizieren.

```
1 student@berlin:~$ echo "Bananen kaufen" > /tmp/notiz
2 student@berlin:~$ cd /tmp/
3 student@berlin:/tmp$ cat notiz
4 Bananen kaufen
```

Die Ausgabe von `notiz` klappt nur, wenn man im richtigen Verzeichnis ist. Befindet man sich im falschen Ordner, in welchem es keine Datei namens `notiz` gibt, wird ein Fehler ausgegeben:

```
1 student@berlin:/tmp$ cd /etc/
2 student@berlin:/etc$ cat notiz
3 cat: notiz: No such file or directory
4 student@berlin:/etc$ cat /tmp/notiz
5 Bananen kaufen
```

Im Gegensatz zum Navigieren mit einem grafischen Programm sieht man nicht auf Anhieb, welchen Inhalt ein Verzeichnis hat und muss erst mittels `ls` danach fragen:

```
1 student@berlin:~$ cd /tmp/
2 student@berlin:/tmp$ ls
3 notiz
```

---

<sup>7</sup><https://www.gnu.org/software/bash/manual/bashref.html>

## 6 Wo liegt welcher Inhalt?

Für den Linux-Einsteiger mag das sehr verwirrend sein. „Woher weiß ich denn, an welchem Ort eine bestimmte Datei liegt?“ ist eine berechtigte Frage. Glücklicherweise ist die Ordner-Struktur aller Linux-Distributionen sehr ähnlich aufgebaut und die Verzeichnisse haben alle einen speziellen Zweck. Dieser „Stadtplan“ prägt sich mit der Zeit ein und hilft, Dateien mit bekanntem Zweck zu finden.

Beispiel: Nach der Installation eines neuen (fiktiven) Programmes *superfoo* sollen dessen Einstellungen angepasst werden. Es ist bekannt, dass systemweite Konfigurationsdateien in `/etc` liegen. Deshalb ist es sehr wahrscheinlich, dass die Konfigurationsdatei von *superfoo* im Ordner `/etc/superfoo` oder der Datei `/etc/superfoo.conf` enthalten ist.

Sofern es Einstellungen gibt, die für jeden Benutzer des System unterschiedlich sind, so liegen sie vermutlich im entsprechenden Benutzer-Verzeichnis, wie etwa `/home/student/.superfoo`.

Wurde ein USB-Stick eingesteckt und automatisch eingehängt<sup>8</sup>, so ist er sehr wahrscheinlich in einem Unterordner von `/media` verfügbar.

Diese Grafik<sup>9</sup> gibt einen Überblick über die Zwecke der verschiedenen Ordner:

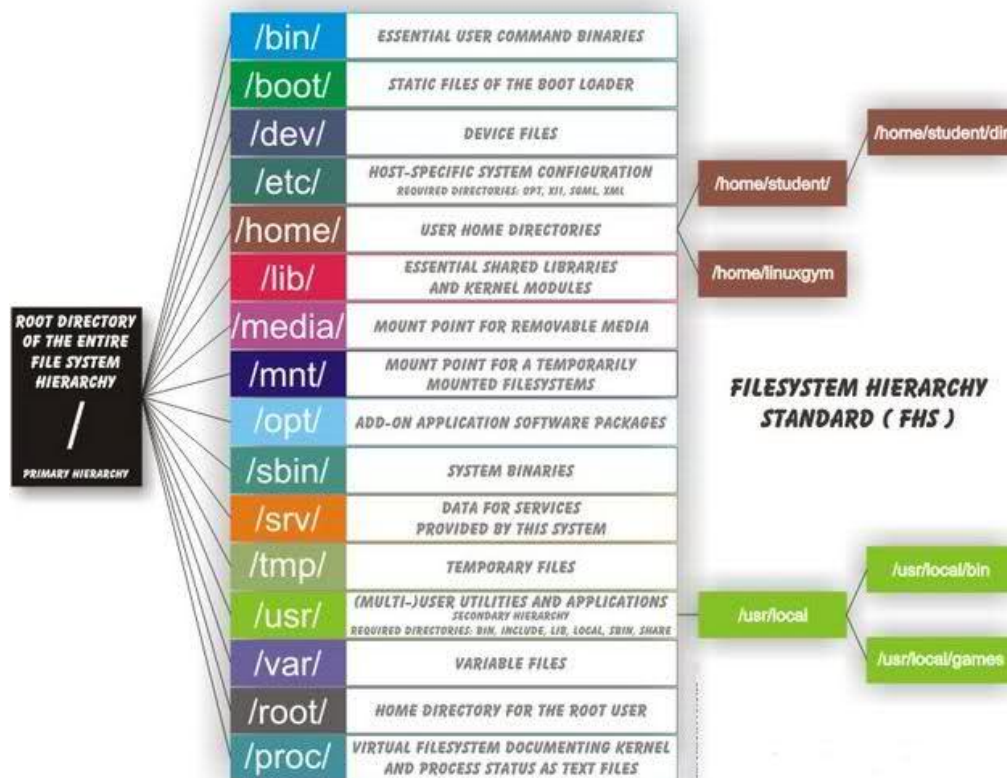


Abbildung 1: Der Verzeichnis-Baum von Linux

<sup>8</sup>„ge-mounted“, also in die Ordner-Struktur eingebunden

<sup>9</sup>*Linux Directory Structure and Important Files Paths Explained* Avishek Kumar, Tecmint, <http://www.tecmint.com/linux-directory-structure-and-important-files-paths-explained/>

## 7 Benutzung von Programmen

Sollte einmal unklar sein, wie ein Programm benutzt wird oder ist ein Parameter unbekannt, kann man die *man page* (kurz für manual page) zurate ziehen. Diese Dokumentation ist standardmäßig für viele Programme installiert und kann mittels `man programmame` aufgerufen werden. Komplexe Programme teilen ihre man pages oft auf und verweisen ganz unten unter „SEE ALSO“ auf weitere man pages.

```

FIND(1)                                General Commands Manual                                FIND(1)

NAME
    find - search for files in a directory hierarchy

SYNOPSIS
    find [-H] [-L] [-P] [-D debugopts] [-Olevel] [path...]
    [expression]

DESCRIPTION
    This manual page documents the GNU version of find.  GNU
    find searches the directory tree rooted at each given file
    name by evaluating the given expression from left to right,
    according to the rules of precedence (see section OPERA-
    TORS), until the outcome is known (the left hand side is
    Manual page find(1) line 1 (press h for help or q to quit)
  
```

Abbildung 2: Die man page des Befehls *find*

Falls der Name eines Programmes nicht bekannt, kann mittels *apropos* gesucht werden. Hier wurde nach dem Begriff „search“ gesucht. Das Ergebnis sind diverse Programme, die mit Suchen zu tun haben.

```

student@PC016:~$ apropos search
apropos (1)          - search the manual page names and descriptions
badblocks (8)        - search a device for bad blocks
bsearch (3)          - binary search of a sorted array
bzipgrep (1)         - search possibly bzip2 compressed files for a regular e...
bzfgrep (1)          - search possibly bzip2 compressed files for a regular e...
bzgrep (1)           - search possibly bzip2 compressed files for a regular e...
find (1)             - search for files in a directory hierarchy
hsearch (3)          - hash table management
hsearch_r (3)        - hash table management
lfind (3)            - linear search of an array
lsearch (3)          - linear search of an array
lzgrep (1)           - search compressed files for a regular expression
lzfgrep (1)          - search compressed files for a regular expression
lzgrep (1)           - search compressed files for a regular expression
  
```

Abbildung 3: apropos suche nach dem Begriff „search“

## 8 Aufgaben

Notieren Sie zu jeder Teilaufgabe die benutzten Befehle und die entsprechenden Ausgaben.

### 8.1 Vorbereitung

Um die für den Versuch notwendigen Dateisysteme zu erhalten, öffnen Sie eine Konsole mittels Tastenkombination Strg-Alt-t oder per Klick auf das Icon im Startmenü und führen Sie den folgenden Befehl aus.

```
1 sudo apt install gruppa-thema-linuxbasics
```

Im Home-Verzeichnis befindet sich nun ein Ordner namens *linux\_basics*. Dieser erhält alle für die folgenden Aufgaben notwendigen Ressourcen.

### 8.2 Navigation im Dateisystem

Bearbeiten Sie die folgenden Aufgaben in der Konsole.

1. In welchem Verzeichnis befinden Sie sich? Mit welchem Befehl finden Sie dies heraus?
2. Wechseln Sie in das Eltern-Verzeichnis. Das heißt, eine Hierarchieebene nach oben. In welchem Verzeichnis befinden Sie sich nun?
3. Geben Sie den Inhalt des Verzeichnisses aus. Gibt es noch andere Verzeichnisse neben dem, in welchem Sie vorher waren?
4. Wechseln Sie in das Verzeichnis `/etc/ssh`. Geben Sie anschließend den Inhalt dieses Verzeichnisses aus.
5. Wechseln Sie nun mittels **relativer** Pfad-Angaben in das Verzeichnis `/home/student/Praktikum/linux_basics`. Benutzen Sie dabei **keine absolute** Pfadangabe. Kann man dies mit einem einzigen Befehl bewerkstelligen?

### 8.3 Dateien bewegen

1. Wechseln Sie in das Verzeichnis `/home/student/Praktikum/linux_basics/task_4`. Dort finden Sie zwei Ordner namens `unsorted` und `sorted`. Ihre Aufgabe ist es nun, alle Dateien, die auf „kuchen“ enden, in den Ordner `sorted/kuchen` zu verschieben und ebenso alle auf „torte“ endenden in `sorted/torte`. Den Rest verschieben Sie in `sorted/sonstiges`. Benutzen Sie dafür den Asterik-Operator (\*) der Bash. Dieser erweitert Dateinamen, z.B. wird „mv DSC\_\*.jpg ~/Pictures“ alle Dateien, die mit „DSC\_“ beginnen und auf „.jpg“ enden in den Ordner Pictures in ihrem Benutzerverzeichnis<sup>10</sup>. Sie kommen dann mit drei Befehlen aus.
2. Benennen Sie das Verzeichnis `sorted` um in `sortiert`. Auch das geht mittels `mv`.
3. Bewegen Sie das Verzeichnis `sortiert` in das Verzeichnis `/tmp`.
4. Geben Sie den Inhalt aller Ordner in `/tmp/sortiert` aus. Auch das geht in einem Befehl mit Hilfe des Asterisks. Die Ausgabe soll als detaillierte Listenansicht passieren, also pro Datei eine Zeile.

---

<sup>10</sup>**Hinweis:** setzen Sie \* nur in sicheren Umgebungen ein, da es zu Sicherheitsrisiken führen kann.



## 8.4 Finden von Dateien

In dem Verzeichnis `/home/student/Praktikum/linux_basics/task_2` befinden sich viele tausend Verzeichnissen und Dateien mit zufällig gewählten Namen und Inhalten. In diesem Chaos ist es fast unmöglich, eine Bestimmte Datei per Hand zu finden. Die Inhalte können Text oder Binärdaten sein. Ihre Aufgabe ist es nun, die unten beschriebenen Dateien zu finden. Notieren Sie sich zu jeder den Befehl und den relativen Pfad, an welcher die Datei zu finden ist. Der Befehl soll so genau wie möglich sein, sodass maximal zehn Ergebnisse pro Aufruf ausgegeben werden. Es sollen nur die Programme *grep* und *find* benutzt werden.

*Hinweis: Markieren Sie Text in der Konsole und drücken Sie STRG-SHIFT-C um ihn in die Zwischenablage zu kopieren. STRG-SHIFT-V fügt in die Konsole ein.*

1. Die Datei enthält die Zeichenfolge „interblag“ in Kleinbuchstaben (ohne Anführungszeichen).
2. Die Datei enthält die Zeichenfolge „world wide blogosphere“ in einer Mischung aus Groß- und Kleinbuchstaben. Das heißt, die gesuchte Zeichenfolge könnte etwa „WoRLD wiDE BLOGosphere“ lauten.
3. Die Datei enthält eine Email-Adresse der Form `vorname.nachname@provider.domain`
4. Die Datei enthält Text auf Französisch. Hinweis: das Zeichen `é` ist sehr häufig im Französischen. *Um Probleme mit dem Encoding in Latex zu vermeiden, ist es ratsam, einen Screenshot des Ergebnisses zu benutzen.*
5. Der Dateiname lautet `nobody_will_find_this`.
6. Der Dateiname lautet `important` und die Dateigröße beträgt exakt 21 Byte.

## 8.5 Pipelining

Wechseln Sie in der Ordner `/home/student/Praktikum/linux_basics/task_3`. Ihr Ziel ist es, mehrere Zahlen-Serien aufzusummieren und die Parität der Summen zu entscheiden.

1. Dort finden Sie eine Datei namens `numbers`. Geben Sie die Datei auf der Konsole mittels *cat* aus. Was enthält sie?
2. Das Skript `sum.py` liest iterativ Zeilen mit Zahlen ein und gibt deren Summe aus. Es endet bei Eingabe einer Leerzeile. Führen Sie das Skript zunächst aus und berechnen Sie beispielhaft drei Summen. Da das Skript nicht im Such-Pfad für ausführbare Programme befindet, müssen Sie dem Programm ein `./` voran stellen, also `./sum.py`.
3. Das Skript `is_even.py` funktioniert ähnlich. Für jede eingegebene Zahl entscheidet es zwischen gerade und ungerade. Führen Sie das Skript aus und überprüfen Sie seine Funktionsweise mit drei verschiedenen Zahlen.
4. Stellen Sie nun die Eingabe des Summenskripts mittels Pipelining bereit. Nutzen Sie dazu den Pipe-Operator `cat numbers | ./sum.py`.
5. Die Ausgabe aus dem vorherigen Schritt können Sie nun wiederum weiterverarbeiten. Um das Ziel zu erreichen, können Sie einfach in gleicher Art und Weise das Paritätsskript anhängen. Hierbei ist es hilfreich, dass Shells Buch führen. Mit den Pfeiltasten können Sie von unten nach oben die voragegangenen Befehle durchschreiten. Mit **Strg+R** ist sogar eine Suche möglich. Sie befinden sich dann an der Position des gefundenen Befehls und die Pfeiltasten zeigen auf die jeweiligen Befehle davor oder danach. Um das nächste Suchergebnis zu erhalten, drücken Sie erneut **Strg+R**. Für ein besseres Verständnis ist der Datenfluß der Zielfunktion hier als Diagramm dargestellt:
6. Nun, da Sie die geraden von den ungeraden Summen unterscheiden können, sollen Sie nur die

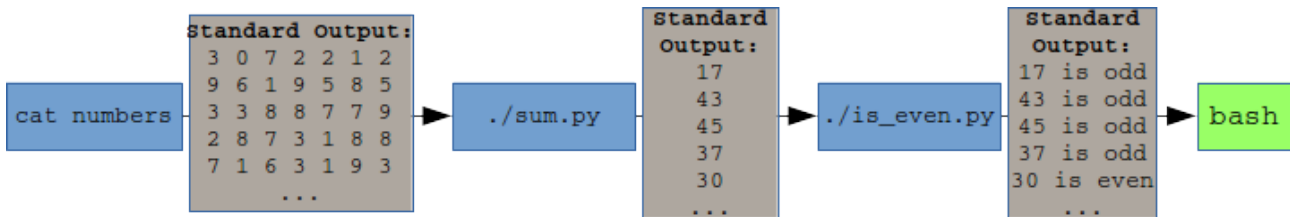


Abbildung 4: Die Ein- und Ausgaben der Befehle.

ungeraden davon ausgeben. Ändern Sie ihren Befehl so ab, dass nur Zeilen ausgegeben werden, in denen das Wort „odd“ vorkommt. Pipen sie dazu die Ausgabe Ihres derzeitigen Befehls in einen Aufruf von *grep* mit passenden Parametern.

7. Da es häufig vorkommt, dass man den Inhalt einer Datei in den Aufruf eines Programmes pipen will, gibt es dafür eine kürzere Schreibweise. Die Ihnen bekannte Möglichkeit ist:

```
1 $ cat datei | programm
```

Dies ist äquivalent zu :

```
1 $ programm < datei
```

In beiden Fällen wird der Inhalt der Datei in die Standard-Eingabe des laufenden Programmes geschrieben.

**Aber Achtung:** folgendes funktioniert nicht.

```
1 $ programm2 < programm1 < datei
```

Um so etwas umzusetzen, muss man die Pipeline so strukturieren:

```
1 $ programm1 < datei | programm2
```

## 8.6 Ein praktischer Anwendungsfall

Sie benötigen zufällige, Hex-Tokens einer Länge von 16 Zeichen. Konstruieren Sie eine Kommandozeile aus der Verkettung von Befehlen mittels Pipelining. Bedenken sie hierbei, dass ein Byte durch 2 „Hexadezimalziffern“ dargestellt wird. Anders gesagt stellt eine „Hexadezimalziffer“ ein Nibble, ein halbes Byte, dar.

Sie können dafür die spezielle Datei `/dev/urandom` benutzen, welche eine nicht endende Menge an zufälligen Binärdaten enthält.

Mit *head* können Sie den ersten Teil einer Eingabe ausgeben lassen, z.B. die ersten 10 Zeilen oder die ersten 5 Bytes. Benutzen Sie einen Parameter, um die Anzahl der auszugebenden Bytes festzulegen.

Mit *xxd* können Sie Binärdaten in Hexdaten umwandeln. Benutzen Sie das Programm im *plain text mode* (siehe man page).

Schreiben Sie den Token mittels Weiterleitung der Ausgabe durch `>` in die Datei `/tmp/token`. Der `>` Operator überschreibt die Datei und ist im Allgemeinen vielseitig<sup>11</sup> und ermöglicht z.B. StdOut und StdErr zusammenzuführen oder StdErr zu ignorieren.

<sup>11</sup><https://devhints.io/bash#redirection>, dieses cheatsheet ist auch für die Zusatzaufgabe sehr hilfreich

## 8.7 Daten im Netzwerk übertragen

In manchen Situationen sitzen Sie nebeneinander und müssen Daten von einem Gerät zum anderen bewegen. Die Daten können alles von URLs sein, die man nicht abtippen will, bis hin zu Gigabyte-großen Dateien, die kein File-Hoster akzeptiert. Möglichkeiten der Datenübertragung gibt es viele: USB-Sticks, Bluetooth, als Email-Anhang, Dropbox, Facebook, Floppy Disks, Lochkarten, Kugelschreiber etc.

Aber was, wenn all diese Möglichkeiten nicht umsetzbar sind? Was ist die einfachste Möglichkeit, Daten im Netzwerk zu übertragen? Die Antwort ist netcat.

*Arbeiten Sie für diesen Aufgabenteil mit ihrer Nachbargruppe zusammen. Sollte diese noch die vorherigen Aufgaben bearbeiten, versuchen sie sich derweil an Aufgabe 8.9.*

Suchen Sie eine interessante URL und übertragen Sie sie per Netcat:

- Empfänger: Rufen Sie netcat im Server-Modus auf und lauschen Sie auf Port 7777. Notieren Sie sich die Ausgabe.
- Absender: Geben Sie die URL mittels *echo* auf der Kommandozeile aus. Führen Sie den Befehl ein zweites Mal aus und pipen Sie die Ausgabe in netcat. Übergeben sie netcat die IP-Adresse ihrer Nachbargruppe und den Port 7777 als Parameter.
- Wechseln Sie die Rollen.

Übertragen Sie den Hex-Token, welchen Sie in den vorigen Aufgaben erzeugt haben, auf den anderen PC.

- Empfänger: Rufen Sie netcat im Server-Modus auf und lauschen Sie auf Port 7777. Leiten Sie die Ausgabe in die Datei `/tmp/neighbor_token`.
- Absender: Versenden Sie die Datei mittels NetCat an die IP-Adresse ihrer Nachbargruppe auf Port 7777. Leiten Sie dafür die Datei `/tmp/token` mit `<` in den Aufruf von Netcat
- Wechseln Sie die Rollen.

Geben Sie im Bericht an, mit welcher Gruppe Sie zusammengearbeitet haben, sowie die beiden benutzten Tokens.

## 8.8 Never have I ever...

Nun betrachten Sie einige Befehle, die potentiell Schaden an ihrem Computer hervorrufen können. Erklären Sie die Befehle und deren Auswirkungen. Geben sie wie immer alle genutzten Quellen an.

**Führen Sie die Befehle nicht aus. Das Ausführen der Befehle führt zum Nicht-Bestehen des Versuches.**

1. `rm -rf /`

Warum ist es bei diesem Befehl besonders gefährlich, wenn man root-Rechte hat?

2. `rm -f *`

Hier zeigt sich die Gefährlichkeit des `*`-Operators. Wie unterscheidet sich der Befehl zum vorigen?

3. `:(){ :|: & };:`

Eine *Fork-Bomb* für die Bash. Wie funktioniert sie? Hier noch einmal in sauberer Formatierung:

```
1 :() {
2   :|: &
3 };
4 :
```

4. `dd if=/dev/zero of=/dev/sda`

In welchen Fällen kann dieser Befehl produktiv genutzt werden?

5. `mv ~ /dev/null`

Wohin werden die Dateien verschoben?

6. `wget http://example.com/something -O - | sh`

Was passiert, wenn das erhaltene Script bösartig ist? Kann dieser Befehl permanenten Schaden hinterlassen?

7. `chmod -R 777 /`

Welche Auswirkungen hat dieser Befehl auf die Privatsphäre bei Systemen mit mehreren Nutzern?

## 8.9 Zusatzaufgabe: Ein Bash-Script reparieren

Im Versuchs-Ordner finden Sie ein Bash-Script namens `broken.sh`, welches zahlreiche Programmierfehler enthält. Jeder Befehl wurde einzeln getestet und in den Kommentaren mit möglichen Fehlerursachen versehen.

Ihre Aufgabe ist es, die Fehler zu beheben und das Skript zu reparieren. Den Zweck des Skriptes können Sie anhand der vorhandenen Kommentare nachvollziehen. Das reparierte Skript gibt einen System-Report aus. Sie können das Skript zur Bearbeitung der Aufgabe im Text-Editor öffnen

```

1  #!/bin/bash
2
3  #TODO: Dear future self, I hope you can fix this script. Sincerly, past-me
4
5  #####
6  # This script gathers system information #
7  #####
8
9  # It can only run as root
10 #TODO: how to negate this -eq. The script shall exit if it is NOT root.
11 if [[ $EUID -eq 0 ]]; then
12     echo "The script must be run as root" 1>&2
13     exit 1
14 fi
15
16 #TODO: If the check above works fine, it's about time to remove this line:
17 exit 1
18
19 # The temporary folder to store collected information
20 TMPDIR="/tmp/sysreport"
21 # The name of the report text file
22 REPORTNAME="report-`hostname`-`date +%F-%T`"
23 # The full path to the report
24 REPORT="${TMPDIR}/${REPORTNAME}"
25 # The full path to the archive file
26 ARCHIVE="$HOME/reports.tar"
27
28 # Assure that the temporary folder exists
29 # TODO: make work if the folder actually exists... google says something about 'parents'
30 mkdir $TMPDIR
31
32 # Start the system report:
33 echo "--- System report of `hostname` ---" >> $REPORT
34
35 # Hardware: Get info about the built-in CPUs:
36 #TODO: learn how to escape those 'or'-operators in double quotes
37 echo "-- CPU info: --" >> $REPORT
38 grep "model name|processor|cpu MHz" /proc/cpuinfo >> $REPORT
39 echo "" >> $REPORT
40
41 # Hardware: The total amount and the free amount of memory:

```

```
42 #TODO: oh no! This totally does not look like two lines!
43 echo "-- Memory info: --" >> $REPORT
44 head -c2 /proc/meminfo >> $REPORT
45 echo "" >> $REPORT
46
47 # Network: Rules of the current firewall settings
48 #TODO: find correct location of iptables. Where is it?.. hm... ---> whereis <--- it?
49 echo "-- IPtable rules: --" >> $REPORT
50 /etc/bin/share/iptables -L >> $REPORT
51 echo "" >> $REPORT
52
53 # Network: Routing information
54 #TODO: I just copy&pasted this from the internet, but it looks weird...
55 echo "-- Routing table: --" >> $REPORT
56 route | rev >> $REPORT
57 echo "" >> $REPORT
58
59 # Network: Configuration of all network interfaces
60 #TODO: I would like to include all interfaces, even those that are down (inactive)
61 echo "-- Network interfaces --" >> $REPORT
62 ifconfig -s >> $REPORT
63 echo "" >> $REPORT
64
65 # Mounts: Currently mounted devices
66 #TODO: this seems to overwrite the report file... maybe a permission issue?
67 echo "-- Mounted objects --" >> $REPORT
68 cp /proc/mounts $REPORT
69 echo "" >> $REPORT
70
71 # Mounts: Pre-configured mounts
72 #TODO: I can't find the error here...
73 echo "-- Pre-configured mounts --" >> $REPORT
74 cat /etc/fstab >> $REPORT
75 echo "" > $REPORT
76
77 # Show it to the user (works)
78 less $REPORT
79
80 # Check whether the file exists
81 cd $TMPDIR
82 if [ ! -f $ARCHIVE ];
83 then
84     # create the TAR-Archive
85     echo "Creating new archive '$ARCHIVE', adding report"
86     touch $ARCHIVE
87     tar --erstellen -f $ARCHIVE $REPORTNAME
88 else
89     # Add the report to the TAR-Archive
90     echo "Adding report to archive '$ARCHIVE'"
91     tar --anhaengen -f $ARCHIVE $REPORTNAME
92 fi
93
94 # Change the permissions so that everybody can read and write the archive file
95 chmod 999 $ARCHIVE
96
97 #TODO: find out where the archive is located
```

## 9 Kontrollfragen

1. Was ist Linux (ursprüngliche Definition)?
2. Was ist Ubuntu?
3. Was ist die Shell?
4. Was gab es zuerst: die Shell oder grafische Umgebungen?
5. Schließt das Vorhandensein einer grafischen Oberfläche das Benutzen einer Shell aus?
6. Was macht *mv* und wie benutzt man es?
7. Was macht *cp* und wie benutzt man es?
8. Was macht *ls* und wie benutzt man es?
9. Wie navigiert man mit der Shell?
10. Was ist der Unterschied zwischen absoluten und relativen Pfad-Angaben?
11. Was ist Pipelining?
12. Was ist NetCat?
13. Wie finden Sie Informationen über die Benutzung eines Programms?