4.3 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	4 Das Linux-Betriebssystem
Lernziel:	4.3 Wo Daten gespeichert werden
Lektion:	1 von 2

Einführung

Für ein Betriebssystem ist alles Daten. **Für Linux ist alles eine Datei**: **Programme, normale Dateien, Verzeichnisse, Blockgeräte bzw. Block Devices** (z.B. Festplatten), zeichenorientierte Geräte bzw. Character Devices (z.B. Konsolen), Kernelprozesse, Sockets, Partitionen, Links etc. **Die Linux-Verzeichnisstruktur, beginnend mit dem Wurzelverzeichnis** (auch ***root*** genannt und symbolisiert durch `/`), ist eine Sammlung von Dateien mit Daten. Dass alles eine Datei ist, ist ein mächtiges Feature von Linux, da sich so praktisch jeder Bereich des Systems optimieren lässt.

In dieser Lektion werden wir die verschiedenen Orte besprechen, an denen **wichtige Daten gespeichert werden**, wie sie der **Filesystem Hierarchy Standard (FHS)** festlegt. Einige dieser Orte sind **echte Verzeichnisse**, die Daten dauerhaft auf Festplatten speichern, während **andere Pseudodateisysteme sind, die in den Speicher geladen werden** und uns Zugang zu Daten des Kernel-Subsystems wie etwa laufende Prozesse, den Speicherverbrauch, die Hardware-Konfiguration usw. geben. Die in diesen **virtuellen Verzeichnissen** gespeicherten Daten werden von einer Reihe von Befehlen verwendet, die es uns ermöglichen, sie zu überwachen und zu verwalten.

Programme und ihre Konfiguration

Wichtige Daten auf einem Linux-System sind zweifellos seine **Programme und deren Konfigurationsdateien.** Erstere sind ausführbare Dateien mit Anweisungen, die vom Prozessor des Computers ausgeführt werden sollen, während letztere in der Regel Textdokumente sind, die den Betrieb eines Programms steuern**. Ausführbare Dateien** können entweder **Binärdateien oder Textdateien** sein. Ausführbare Textdateien werden als Skripte bezeichnet. **Konfigurationsdaten** unter Linux werden traditionell auch in **Textdateien gespeichert**, obwohl es verschiedene Arten der Darstellung von Konfigurationsdaten gibt.

Wo Binärdateien gespeichert sind

Wie alle anderen Dateien liegen ausführbare Dateien in Verzeichnissen unterhalb von `/`. Genauer gesagt, werden **Programme über eine dreistufige Struktur verteilt**: Die **erste Schicht (`/`) enthält Programme, die im Einzelbenutzermodus notwendig** sein können, die **zweite Schicht (`/usr`) enthält die meisten Mehrbenutzerprogramme** und die **dritte Schicht (`/usr/local`) wird verwendet, um**

Software zu speichern, die nicht von der Distribution bereitgestellt und lokal kompiliert wurde.

Typische Orte für Programme sind:

`/sbin`
: enthält wichtige **Binärdateien** für die Systemadministration wie `parted` oder `ip`.

`/bin`
: enthält **essentielle Binärdateien** für alle Benutzer wie `ls`, `mv` oder `mkdir`.

`/usr/sbin`
: enthält Binärdateien für die **Systemadministration** wie `deluser` oder `groupadd`.

`/usr/bin`
: enthält die meisten **ausführbaren Dateien —** wie `free`, `pstree`, `sudo` oder `man` --, die von allen Benutzern verwendet werden können.

`/usr/local/sbin`
: wird verwendet, um **lokal installierte Programme** für die Systemadministration zu speichern, die **nicht vom Paketmanager des Systems verwaltet** werden.

`/usr/local/bin`
: dient dem gleichen Zweck wie `/usr/local/sbin`, jedoch für **normale Benutzerprogramme.**

Einige Distributionen sind dazu übergegangen, `/bin` und `/sbin` durch **symbolische Links** zu `/usr/bin` und `/usr/sbin` zu ersetzen.

Note Das Verzeichnis `/opt` wird manchmal zur Ablage optionaler Anwendungen von Drittanbietern verwendet.

Abgesehen von diesen Verzeichnissen können normale Benutzer ihre eigenen Programme in den folgenden haben:

- `/home/$USER/bin`
- `/home/$USER/.local/bin`

Tip Sie sehen, aus welchen Verzeichnissen heraus Sie Binärdateien ausführen können, indem Sie den Variableninhalt von **`PATH`** mit dem Kommando `echo $PATH` ausgeben. Weitere Informationen zu `PATH` finden Sie in den Lektionen über Variablen und Shell-Anpassung.

Den Standort eines Programms liefert der Befehl `which`:

```
$ which git
/usr/bin/git
```

Wo Konfigurationsdateien liegen

Das Verzeichnis `/etc`

In den frühen Tagen von Unix gab es für jeden Datentyp einen Ordner, wie z.B. **`/bin` für Binärdateien** und **`/boot`** für **den/die Kernel**. **`/etc` (für "et cetera")** wurde als **Catch-All-Verzeichnis angelegt**, um alle Dateien zu speichern, die nicht zu den anderen Kategorien gehörten, und die meisten dieser Dateien waren eben

Konfigurationsdateien. Mit der **Zeit wurden immer mehr Konfigurationsdateien** hinzugefügt, so dass **`/etc` zum Hauptordner für Konfigurationsdateien** von Programmen wurde. Wie bereits erwähnt, ist eine Konfigurationsdatei in der Regel eine **lokale Textdatei** (im Gegensatz zu einer binären), die den Betrieb eines Programms steuert.

In `/etc` finden wir verschiedene Muster zur Benennung von Konfigurationsdateien:

- **Dateien mit einer *ad hoc* Erweiterung oder gar keiner Erweiterung, z.B**.

  `group`
  Systemgruppen-Datenbank

  `hostname`
  Name des Host-Computers

  `hosts`
  Liste der IP-Adressen und deren Hostnamen-Übersetzungen

  `passwd`
  Systembenutzer-Datenbank — bestehend aus sieben Feldern, die durch Doppelpunkte getrennt sind und Informationen über den Benutzer liefern

  `profile`
  Systemweite Konfigurationsdatei für Bash

  `shadow`
  Verschlüsselte Datei für Benutzerpasswörter

- **Initialisierungsdateien mit der Endung `rc`:**

  `bash.bashrc`
  Systemweite `.bashrc` Datei für interaktive Bash Shells

  `nanorc`
  Beispiel-Initialisierungsdatei für GNU nano (ein einfacher Texteditor, der normalerweise mit jeder Distribution ausgeliefert wird)

- **Dateien mit der Endung `.conf`:**

  `resolv.conf`
  Config-Datei für den Resolver, der den Zugriff auf das Internet Domain Name System (DNS) ermöglicht

  `sysctl.conf`
  Config-Datei zum Setzen von Systemvariablen für den Kernel

- **Verzeichnisse mit dem Suffix `.d`:**

  Einige Programme mit einer eindeutigen Konfigurationsdatei (`*.conf` oder ähnlich) haben sich zu einem **dedizierten Verzeichnis `*.d` entwickelt**, das den Aufbau modularer, robusterer Konfigurationen ermöglicht. Zum Beispiel finden Sie bei der Konfiguration von **logrotate `logrotate.conf`, aber auch `logrotate.d` Verzeichnisse.**

Dieser Ansatz ist besonders nützlich, wenn verschiedene Anwendungen Konfigurationen für denselben spezifischen Dienst benötigen. Wenn beispielsweise ein Webserver-Paket eine logrotate-Konfiguration enthält, kann diese Konfiguration nun in eine eigene Datei im Verzeichnis `logrotate.d` abgelegt werden. Diese Datei kann vom Webserver-Paket aktualisiert werden, ohne die übrige logrotate-Konfiguration zu beeinträchtigen. Ebenso können **Pakete bestimmte Aufgaben hinzufügen**, indem sie Dateien in das Verzeichnis **`/etc/cron.d`** legen, **anstatt `/etc/crontab` zu ändern.**

In Debian — und Debian-Derivaten — wurde dieser Ansatz auf die Liste der vertrauenswürdigen Quellen angewendet, die vom Paketverwaltungswerkzeug `apt` gelesen werden: Abgesehen vom klassischen `/etc/apt/sources.list` finden wir jetzt das Verzeichnis `/etc/apt/sources.list.d`:

```
$ ls /etc/apt/sources*
/etc/apt/sources.list
/etc/apt/sources.list.d:
```

Konfigurationsdateien im Home-Verzeichnis (Dotfiles)

Auf Benutzerebene speichern Programme ihre Konfigurationen und Einstellungen in **versteckten Dateien im Heimatverzeichnis des Benutzers** (dargestellt als `~`), wobei versteckte Dateien mit einem **Punkt (`.`) beginnen** — daher ihr Name: ***Dotfiles***.

Einige dieser Dotfiles sind Bash-Skripte, die die Shell-Sitzung des Benutzers anpassen und ausgelesen werden, sobald sich der Benutzer am System anmeldet:

`.bash_history`
: speichert den Verlauf der Kommandozeile

`.bash_logout`
: enthält Befehle, die beim Verlassen der Login-Shell ausgeführt werden sollen

`.bashrc`
: Initialisierungsskript der Bash für Nicht-Login-Shells

`.profile`
: Initialisierungsskript der Bash für Login-Shells

> **Note** Lesen Sie die Lektion über “Grundlagen der Befehlszeile”, um mehr über Bash und seine Init-Dateien zu erfahren.

Andere benutzerspezifische Programmkonfigurationsdateien werden beim Start der jeweiligen Programme ausgewertet: `.gitconfig`, `.emacs.d`, `.ssh`, `.ssh` etc.

Der Linux-Kernel

Bevor ein Prozess ausgeführt werden kann, muss der Kernel in einen geschützten Speicherbereich geladen werden. Danach löst der **Prozess mit PID `1` (heute meist `systemd`) die Prozesskette aus**, d.h. ein **Prozess startet einen anderen Prozess** usw. Sobald die Prozesse aktiv sind, kann ihnen der Linux-Kernel Ressourcen (Tastatur, Maus, Festplatten, Speicher, Netzwerkschnittstellen usw.) zuweisen.

Note **Vor `systemd` war `/sbin/init` immer der erste Prozess in einem Linux-System als Teil** des *System V Init* System Managers. Tatsächlich finden Sie **`/sbin/init` immer noch, aber in der Regel als Link zu `/lib/systemd/systemd`.**

Wo Kernel gespeichert sind: `/boot`

Der Kernel befindet sich in `/boot` — zusammen mit anderen boot-bezogenen Dateien, von denen die meisten die Teile der Kernel-Versionsnummer im Namen haben (Kernel-Version, Major-Revision, Minor-Revision und Patch-Nummer).

Das Verzeichnis `/boot` enthält die folgenden Dateitypen, deren Namen der jeweiligen Kernelversion entsprechen:

`config-4.9.0-9-amd64`
: **Konfigurationseinstellungen** für den Kernel wie Optionen und Module, die zusammen mit dem Kernel kompiliert wurden.

`initrd.img-4.9.0-9-amd64`
: **Initiales RAM-Disk-Image**, das beim Start hilft, indem es ein temporäres Root-Dateisystem in den Speicher lädt.

`System-map-4.9.0-9-amd64`
: Die **Datei `System-map`** (auf einigen Systemen auch `System.map`) enthält Speicheradressorte für Kernel-Symbolnamen. Jedes Mal, wenn ein Kernel neu gebaut wird, ändert sich der Inhalt der Datei, da die **Speicherorte** unterschiedlich sein können. Der Kernel benutzt diese Datei, um Speicheradressorte für ein bestimmtes Kernel-Symbol nachzuschlagen, oder umgekehrt.

`vmlinuz-4.9.0-9-amd64`
: **Der Kernel selbst** in einem **selbstextrahierenden, platzsparenden, komprimierten Format** (dafür steht das `z` in `vmlinuz`; `vm` steht für virtuellen Speicher (virtual memory) und wurde verwendet, als der Kernel zum ersten Mal Unterstützung für virtuellen Speicher erhielt).

`grub`
: Konfigurationsverzeichnis für den **`grub2` Bootloader.**

Tip Da es sich um ein kritisches Merkmal des Betriebssystems handelt, werden **mehr als ein Kernel und die zugehörigen Dateien in `/boot` aufbewahrt**, falls die Standardversion fehlerhaft wird und wir auf eine **frühere Version zurückgreifen müssen**, um wenigstens das System starten und reparieren zu können.

Das Verzeichnis `/proc`

Das Verzeichnis `/proc` ist eines der sogenannten **virtuellen oder Pseudo-Dateisysteme,** da sein Inhalt **nicht auf die Festplatte geschrieben**, sondern in den **Arbeitsspeicher geladen** wird. Es wird bei jedem Hochfahren des Computers dynamisch gefüllt und spiegelt ständig den **aktuellen Zustand des Systems** wider. `/proc` enthält Informationen über:

- **Laufende Prozesse**
- **Kernelkonfiguration**

- **Systemhardware**

Neben allen Daten zu Prozessen, die wir in der nächsten Lektion sehen werden, **speichert** dieses Verzeichnis **auch Dateien mit Informationen über die Hardware des Systems und die Konfigurationseinstellungen des Kernels**, darunter einige dieser Dateien:

`/proc/cpuinfo`

speichert Informationen über die CPU des Systems:

```
$ cat /proc/cpuinfo
processor: 0
vendor_id: GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
stepping : 10
cpu MHz         : 3696.000
cache size      : 12288 KB
(...)
```

`/proc/cmdline`

speichert die Zeichenketten, die beim Booten an den Kernel übergeben werden:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64 root=UUID=5216e1e4-ae0e-441f-
b8f5-8061c0034c74 ro quiet
```

`/proc/modules`

zeigt die Liste der in den Kernel geladenen Module an:

```
$ cat /proc/modules
nls_utf8 16384 1 - Live 0xffffffffc0644000
isofs 40960 1 - Live 0xffffffffc0635000
udf 90112 0 - Live 0xffffffffc061e000
crc_itu_t 16384 1 udf, Live 0xffffffffc04be000
fuse 98304 3 - Live 0xffffffffc0605000
vboxsf 45056 0 - Live 0xffffffffc05f9000 (O)
joydev 20480 0 - Live 0xffffffffc056e000
vboxguest 327680 5 vboxsf, Live 0xffffffffc05a8000 (O)
hid_generic 16384 0 - Live 0xffffffffc0569000
(...)
```

Das Verzeichnis `/proc/sys`

Dieses Verzeichnis enthält **Kernelkonfigurationseinstellungen in Dateien**, die in Kategorien pro Unterverzeichnis eingeteilt sind:

```
$ ls /proc/sys
abi  debug  dev  fs  kernel  net  user  vm
```

Die meisten dieser Dateien verhalten sich wie ein Schalter und enthalten daher nur einen der beiden **möglichen Werte: `0` oder `1` ("on" oder "off"),** zum Beispiel:

`/proc/sys/net/ipv4/ip_forward`

Der Wert, der unsere Maschine aktiviert oder deaktiviert, um als Router zu fungieren (d.h. Pakete weiterleiten zu können):

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

Es gibt jedoch einige Ausnahmen:

`/proc/sys/kernel/pid_max`

Die maximal zulässige PID:

```
$ cat /proc/sys/kernel/pid_max
32768
```

Warning Seien Sie besonders vorsichtig, wenn Sie die Kernel-Einstellungen ändern, da ein falscher Wert zu einem instabilen System führen kann.

Hardware-Geräte

Denken Sie daran: **Unter Linux "ist alles eine Datei**". Auch Hardware-Geräteinformationen und die eigenen Konfigurationseinstellungen des Kernels werden in speziellen Dateien gespeichert, die sich in virtuellen Verzeichnissen befinden.

Das Verzeichnis `/dev`

Das **Geräteverzeichnis `/dev` ("device") enthält Gerätedateien** (*Nodes* oder Knoten) für alle angeschlossenen Hardware-Geräte. Diese Gerätedateien bilden die Schnittstelle zwischen den Geräten und den sie verwendenden Prozessen. Jede dieser Dateien gehört zu einer von zwei Kategorien:

Blockgeräte oder Block Devices

Sind solche, bei denen **Daten in Blöcken gelesen und geschrieben** werden, die individuell adressiert werden können, z.B. Festplatten (und deren Partitionen, wie `/dev/sda1`), USB-Sticks, CDs, DVDs etc.

Zeichenorientierte Geräte oder Character Devices

Sind solche, bei denen **Daten nacheinander zeichenweise gelesen und geschrieben** werden, z.B. Tastaturen, die Textkonsole (`/dev/console`), serielle Schnittstellen (wie `/dev/ttyS0` und so weiter) etc.

Wenn Sie Gerätedateien auflisten, stellen Sie sicher, dass **Sie `ls` mit dem Schalter `-l` verwenden**, um zwischen den beiden zu unterscheiden. Wir können zum Beispiel nach Festplatten und Partitionen suchen:

```
# ls -l /dev/sd*
brw-rw---- 1 root disk 8, 0 may 25 17:02 /dev/sda
brw-rw---- 1 root disk 8, 1 may 25 17:02 /dev/sda1
brw-rw---- 1 root disk 8, 2 may 25 17:02 /dev/sda2
(...)
```

Oder nach **seriellen Terminals** (TeleTYpewriter):

```
# ls -l /dev/tty*
crw-rw-rw- 1 root tty     5,  0 may 25 17:26 /dev/tty
crw--w---- 1 root tty     4,  0 may 25 17:26 /dev/tty0
crw--w---- 1 root tty     4,  1 may 25 17:26 /dev/tty1
(...)
```

Beachten Sie, wie das erste Zeichen **`b` Block** Devices bzw. **`c` Character** Devices kennzeichnet.

Tip Das Sternchen (`*`) ist ein Globbing-Zeichen, das 0 oder mehr Zeichen repräsentiert. Daher ist es in den obigen Befehlen `ls -l /dev/sd*` und `ls -l /dev/tty*` wichtig. Um mehr über diese Sonderzeichen zu erfahren, lesen Sie die Lektion über Globbing.

Außerdem enthält `/dev` einige spezielle Dateien, die für verschiedene Programmierzwecke sehr nützlich sind:

`/dev/zero`
: stellt so **viele Nullzeichen** wie gewünscht bereit.

`/dev/null`
: auch "**Datenmülleimer**" oder "bit bucket" genannt, verwirft alle Informationen, die dorthin gesendet werden.

`/dev/urandom`
: erzeugt **Pseudozufallszahlen**.

Das Verzeichnis `/sys`

Das sys-Dateisystem (`sysfs`) ist auf `/sys` gemountet. Es wurde mit Kernel 2.6 eingeführt und bedeutete eine große Verbesserung gegenüber `/proc/sys`.

Prozesse müssen mit den Geräten in `/dev` interagieren, und so benötigt der Kernel ein Verzeichnis, das Informationen über diese Hardware-Geräte enthält. Dieses Verzeichnis ist `/sys`, und seine **Daten sind nach Kategorien geordnet**. Um beispielsweise die MAC-Adresse Ihrer Netzwerkkarte (`enp0s3`) zu überprüfen, würden Sie die folgende Datei mit `cat` anzeigen:

```
$ cat /sys/class/net/enp0s3/address
08:00:27:02:b2:74
```

Speicher und Speichertypen

Damit ein Programm ausgeführt werden kann, muss es grundsätzlich in den Speicher geladen werden. Im Allgemeinen beziehen wir uns beim Thema Speicher auf *Random Access Memory* (RAM); verglichen mit mechanischen Festplatten hat er den Vorteil, viel schneller zu sein. Auf der anderen Seite ist er volatil, d.h. wenn der Computer heruntergefahren wird, sind die Daten weg.

Wenn es um Speicher geht, unterscheiden wir **zwei Haupttypen** in einem Linux-System:

Physischen Speicher
: auch **bekannt als *RAM***, hat die Form von Chips, die aus integrierten Schaltungen mit Millionen von Transistoren und Kondensatoren bestehen. Diese wiederum bilden Speicherzellen (der Grundbaustein des Computerspeichers), von denen jeder ein hexadezimaler Code (eine Speicheradresse) zugeordnet ist, so dass er bei Bedarf referenziert werden kann.

Swap
: auch ***Swap Space*** genannt, ist der Teil des virtuellen Speichers, der auf der **Festplatte lebt** und verwendet wird, wenn kein RAM mehr verfügbar ist.

Auf der anderen Seite gibt es das Konzept des *virtuellen Speichers*, eine Abstraktion der Gesamtmenge an nutzbarem, adressierendem Speicher (RAM, aber auch Festplattenspeicher) aus Sicht der Anwendungen.

`free` parst **`/proc/meminfo`** und zeigt die Menge an freiem und verbrauchtem Speicher im System sehr übersichtlich an:

```
$ free
               total        used        free      shared  buff/cache
available
Mem:         4050960     1474960     1482260       96900     1093740
2246372
Swap:        4192252           0     4192252
```

Klären wir die Bedeutung der einzelnen Spalten:

`total`
: Gesamtmenge des installierten physischen und Swap-Speichers.

`used`
: Menge des aktuell verwendeten physischen und Swap-Speichers.

`free`
: Menge des physischen und Swap-Speichers, die aktuell nicht verwendet wird.

`shared`
: Menge des (meist) von `tmpfs` verwendeten physikalischen Speichers.

`buff/cache`
: Menge des physischen Speichers, der aktuell von Kernelpuffern, dem Seitencache und den Slabs verwendet wird.

`available`
: schätzt, wie viel physischer Speicher für neue Prozesse zur Verfügung steht.

Standardmäßig zeigt `free` **Werte in Kibibytes** an, ermöglicht aber eine Vielzahl von Schaltern, um die Ergebnisse in verschiedenen Maßeinheiten anzuzeigen, darunter folgende:

`-b`
: Bytes

`-m`
: Mebibytes

`-g`
: Gibibytes

`-h`
: für Menschen lesbares Format

`-h` ist immer angenehm zu lesen:

```
$ free -h
               total        used        free      shared  buff/cache
available
Mem:           3,9G        1,4G        1,5G         75M        1,0G
2,2G
Swap:          4,0G          0B        4,0G
```

Note **Ein Kibibyte (KiB) entspricht 1.024 Byte**, während ein Kilobyte (KB) 1000 Byte entspricht. Dasselbe gilt jeweils für Mebibytes, Gibibytes, etc.

Geführte Übungen

1. Verwenden Sie den Befehl `which`, um die Position der folgenden Programme herauszufinden und die Tabelle zu vervollständigen:

Programm	`which` Befehl	Pfad zur ausführbaren Datei (Ausgabe)	Benutzer benötigt `root`-Rechte?
`swapon`			
`kill`			
`cut`			
`usermod`			
`cron`			
`ps`			

2. Wo sind die folgenden Dateien zu finden?

Datei	`/etc`	`~`
`.bashrc`		
`bash.bashrc`		
`passwd`		
`.profile`		
`resolv.conf`		
`sysctl.conf`		

3. Erklären Sie die Bedeutung der Zahlenelemente für die Kerneldatei `vmlinuz-4.15.0-50-generic` in `/boot`:

Zahlenelement	Bedeutung
`4`	
`15`	
`0`	
`50`	

4. Welchen Befehl würden Sie verwenden, um alle Festplatten und Partitionen in `/dev` aufzulisten?

Offene Übungen

1. Gerätedateien für Festplatten werden auf der Grundlage der Controller dargestellt, die sie verwenden. Wir haben `/dev/sd*` für Laufwerke mit SCSI (Small Computer System Interface) und SATA (Serial Advanced Technology Attachment) gesehen, aber
    - Wie wurden alte IDE (Integrated Drive Electronics) Laufwerke dargestellt?
    - Und moderne NVMe (Non-Volatile Memory Express) Laufwerke?
2. Werfen Sie einen Blick auf die Datei `/proc/meminfo`. Vergleichen Sie den Inhalt dieser Datei mit der Ausgabe des Befehls `free` und identifizieren Sie, welcher Schlüssel aus `/proc/meminfo` den folgenden Feldern in der Ausgabe von `free` entspricht:

`free` Ausgabe	`/proc/meminfo` Feld
`total`	
`free`	
`shared`	
`buff/cache`	
`available`	

Zusammenfassung

In dieser Lektion haben Sie sich mit dem Speicherort von Programmen und deren Konfigurationsdateien in einem Linux-System vertraut gemacht. Wichtige Fakten, die Sie beachten sollten:

- Grundsätzlich sind Programme in einer dreistufigen Verzeichnisstruktur zu finden: `/`, `/usr` und `/usr/local`. Jede dieser Ebenen kann `bin` und `sbin` Verzeichnisse enthalten.
- Konfigurationsdateien werden in `/etc` und `~` gespeichert.
- Punktdateien sind versteckte Dateien, die mit einem Punkt (`.`) beginnen.

Wir haben auch über den Linux-Kernel gesprochen. Wichtige Fakten sind:

- Für Linux ist alles eine Datei.
- Der Linux-Kernel lebt in `/boot` zusammen mit anderen boot-bezogenen Dateien.
- Damit Prozesse mit der Ausführung beginnen können, muss der Kernel zuerst in einen geschützten Speicherbereich geladen werden.
- Aufgabe des Kernels ist es, Systemressourcen den Prozessen zuzuweisen.
- Das virtuelle (oder Pseudo-)Dateisystem `/proc` speichert wichtige Kernel- und Systemdaten auf flüchtige Weise.

Ebenso haben wir uns mit Hardware-Geräten beschäftigt und folgendes gelernt:

- Das Verzeichnis `/dev` speichert spezielle Dateien (auch bekannt als Knoten) für alle angeschlossenen Hardware-Geräte: *Block Devices* oder *Character Devices*. Die ersten übertragen Daten in Blöcken, letztere zeichenweise.
- Das Verzeichnis `/dev` enthält auch andere spezielle Dateien wie `/dev/zero`, `/dev/null` oder `/dev/urandom`.
- Das Verzeichnis `/sys` speichert Informationen über Hardwaregeräte, die Kategorien zugewiesen sind.

Schließlich haben wir Speicher behandelt und gelernt:

- Ein Programm wird ausgeführt, wenn es in den Speicher geladen wird.
- Was RAM (Random Access Memory) ist.
- Was Swap ist.
- Wie man die Speichernutzung anzeigt.

Befehle, die in dieser Lektion verwendet wurden:

`cat`
: Dateiinhalt verketten/ausgeben.

`free`
: Zeigt die Menge an freiem und verbrauchtem Speicher im System an.

`ls`
: Listet Verzeichnisinhalte auf.

`which`
: Zeigt den Standort des Programms an.