Linux Essentials

Michael Dienert

29. September 2019

Inhaltsverzeichnis

I Vorwort

Dieses Skriptum enthält die Lernziele (*Objectives*) des *Linux Essentials*-Lehrgangs des LPI (Linux Professional Institute).

Der LinuxEssentials-Kurs ist eine Ergänzung zu den anspruchsvollen Zertifizierungen LPIC-1 und LPIC-2 des LPI und vom Schwierigkeitsgrad unterhalb dieser beiden angesetzt. Er richtet sich vor allem an Schüler und Studenten.

II Lehrmaterial

Ein kostenloses PDF-Buch zu LinuxEssentials wurde von der Firma *Linup Front GmbH* erstellt. Leider hat die Firma zum 31. Dezember 2015 alle Aktivitäten eingestellt, das Buch ist jedoch noch hier erhältlich:

https://www.tuxcademy.org/download/de/lxes/lxes-de-manual.pdf

Weiteres Lehrmaterial ist das Dokument, das Sie gerade lesen. Die Kapitel entsprechen genau den auf der LPI-Seite veröffentlichten Ojectives. Diese wurden um Erläuterungen, Beispiele und Übungen ergänzt.

1 The Linux Community and a Career in Open Source

1.1 Linux Evolution and Popular Operating Systems

Weight: 2

Key Knowledge Areas:

- Open Source Philosophy
- Distributions

- Embedded Systems

The following is a partial list of the used files, terms and utilities:

- Android
- Debian, Ubuntu (LTS)
- CentOS, openSUSE, Red Hat
- Linux Mint, Scientific Linux

1.1.1 Open Source Philosophy

1.1.2 Distributions - Distributionen

Wie weiter unten erklärt werden wird, müssen die Quelltexte aller OpenSource Programme veröffentlicht werden.
Die Standardprogrammiersprache ist dabei C/C++ mit dem Compiler `gcc`. Dabei bedeutete `gcc` ursprünglich **GNU C-Compiler**. Jetzt steht es für **GNU Compiler Collection**, da `gcc` inzwischen das *Frontend* zu einem ganzen Zoo von Compilern (C, C++, Java, Fortran, Ada, Pascal, ...) ist.
Nun kann sich also jeder ein GNU/Linux Betriebsystem bauen, wenn er einen `gcc` hat und sich aus dem Internet alle gewünschten Quellpakete besorgt hat.
Das ist natürlich sehr mühsam und verlangt einiges an Hintergrundwissen. Aus diesem Grund gibt es Programmsammlungen mit fertig compilierten Programmen, sog. *Binärpaketen*. Derartige Programmsammlungen nennt man *Distributionen*.
Verbreitete Distributionen in Deutschland sind z.B.:

1. Knoppix
2. (K)(X)Ubuntu / Debian
3. RedHat
4. SUSE

Weltweit gibt es mehrere Hundert Distributionen, die zum Teil länderspezifisch sind. China hatte bis 2014 z.B. seine eigene Red Flag Distribution. Dort wird GNU/Linux anstelle von Wintendo in öffentlichen Einrichtungen verwendet. RedFlag Linux ist inzwischen aber eingestampft worden.

1.1.3 Android

Android ist keine echte Linux-Distribution, aber es hat einen Linux-Kernel und ist den sog. Embedded Linux - Distributionen (1.1.4) sehr ähnlich.

1.1.4 Embedded Linux

Ebedded Distributionen sind *massgeschneiderte* Distributionen, die als Firmware auf Geräten wie Routern, Messsgeräten, Audio/Video-Geräten usw. laufen.

1.1.5 Debian, Ubuntu und Raspbian

Debian ist eine der freiesten (s.u.) und einflussreichsten Distributionen. Das Debian Projekt wurde 1993 von Ian Murdock gestartet. Aktuelle Version ist Debian GNU/Linux 8 “Jessie”.

Debian Distributionen gibt es für eine grosse Zahl von Hardware-Architekturen (ARM, i386, AMD64, SPARC, MIPS, PowerPC).

Ubuntu ist die Distribution der Firma Canotix, die von Mark Shuttleworth gegründet wurde. Ubuntu basiert auf Debian, ist also ein Debian-Derivat.

Raspbian ist eine Debian-Distribution, die speziell auf den Einplatinen-Minirechner *RaspberryPI* mit ARM-Architektur zugeschnitten worden ist.

Besonderheit von Debian und aller seiner Derivate ist das vorzügliche System, mit dem die Binärpakete der Distributionen verwaltet werden.

1.1.6 Red Hat, CentOS, SUSE

RedHat ist eine US-Amerikanische Distribution und existiert wie Debian auch seit etwa 1993. Inzwischen konzentriert sich RedHat ausschliesslich auf Unternehmenskunden. Eine Version von RedHat kann man ausschliesslich mit Supportverträgen erwerben.

Für Privatanwender wurde die freie Version *Fedora* abgespalten.

Ebenfalls auf RedHat basiert die Version *CentOS*

SUSE-Linux bzw. openSUSE ist eine Distribution, die seit 1996 existiert. Besonderes Merkmal von openSUSE ist das Konfigurationsprogramm YaST (Yet another Setup Tool). Die Distribution war in Deutschland sehr verbreitet, da die zugehörige Firma SuSE (Gesellschaft für Software- und Systementwicklung) ursprünlich ein Nürnberger Unternehmen war. Inzwischen ist SUSE LLC ein amerikanisches Unternehmen und mit der Tochtergesellschaft SUSE Linux GmbH.

1.1.7 Linux Mint, Scientific Linux

1.2 Major Open Source Applications

Weight: 2
Key Knowledge Areas:

- Desktop Applications
- Server Applications
- Development Languages
- Package Management Tools and repositories

Terms and Utilities:

- OpenOffice.org, LibreOffice, Thunderbird, Firefox, GIMP
- Apache HTTPD, NGINX, MySQL, NFS, Samba
- C, Java, Perl, shell, Python
- `dpkg, apt-get, rpm, yum`

1.3 Understanding Open Source Software and Licensing

Weight: 1
Key Knowledge Areas:

- Licensing
- Free Software Foundation (FSF), Open Source Initiative (OSI)

Terms and Utilities:

- GPL, BSD, Creative Commons
- Free Software, Open Source Software, FOSS, FLOSS
- Open Source business models

1.3.1 Free Software

Die Idee der *Free Software* also *freier* Software, wurde von Richard M. Stallman (RMS) in den 1980er Jahren formuliert.
Dabei wird das Wort *frei* oft falsch verstanden: Es hat hier dieselbe Bedeutung wie *frei* in *freier Rede* und **nicht** *frei* im Sinne von *Freibier*!
Freie Software muss nach R. Stallman 4 Bedingungen erfüllen:

1. freie Benutzung der Software
2. die uneingeschränkte Möglichkeit, die Software zu studieren (d.h. den Quelltext lesen zu können)
3. die Freiheit, die Software ändern zu dürfen
4. die Software uneingeschränkt kopieren und kostenlos weiterzugeben zu dürfen

Er leitet diese Bedingungen aus dem *Solidaritätsprinzip* ab.
Einhergehend mit diesen 4 Forderungen, wurde in den 1980er die *GNU General Public License* (GNU GPL) entwickelt. Das ist eine Softwarelizenz, die Freie Software davor schützen soll, unfrei zu werden.
Die GNU GPL kann hier eingesehen werden:

```
http://www.gnu.org/licenses/gpl.html
```

Die GPL enthält im Kern die obengenannten 4 Bedingungen für *Freie Software*. Besondere Bedeutung hat dabei die Freiheit, die Software studieren zu können: dies setzt voraus, dass auch die Quelltexte der Software offengelegt werden müssen.
Keine Aussagen mach die GNU GPL dagegen zur Frage, ob man Geld für Freie Software verlangen darf. Daraus folgt das Recht des **Autors** der Software, einen *beliebigen Betrag* für Kopien der Binärdateien und/oder der Quellen verlangen.
Er hat aber nicht das Recht Geld dafür zu verlangen, dass andere Leute Kopien seiner Software verwenden und weiterverteilen!

Jeder hat das Recht, Freie Software zu ändern. Solange man die dadurch entstandene Version privat verwendet, müssen diese Änderungen nicht öffentlich gemacht werden! Das gilt auch für grosse Firmen und Organisationen.

Wird die geänderte Software aber an die Allgemeinheit weitergegeben, muss diese wiederum unter der GNU GPL stehen, was natürlich die Offenlegung des Quelltextes erzwingt.

Free Software Foundation

Die Free Software Foundation (Stiftung für Freie Software) wurde 1985 von Richard Stallman gegründet. Ziel der Stiftung ist die Verbreitung von freier Software und die Förderung des GNU-Projekts.

GNU - GNU's Not Unix

Das GNU-Projekt hat sich zur Aufgabe gemacht, ein freies, UNIX-ähnliches Betriebssystem zu entwickeln.

UNIX ist ein Mehrbenutzer- und Multitasking-System, das ein hierarchisch, baumförmiges Dateisystem hat. Es wird seit 1969 entwickelt!

GNU und Linux

Linux selbst ist strenggenommen nur der sog. *Kernel*. Der Kernel sorgt dafür, dass die Hardware über Treibersoftware nach aussen auf eine standardisierte Softwareschnittstelle abgebildet wird (Schichtenmodell).

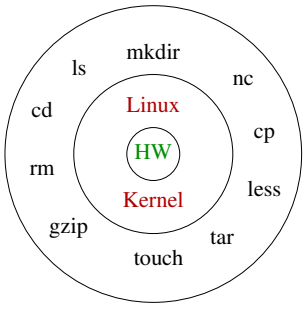

Abbildung 1: Schichtenmodell von Kernel und Anwendungsprogrammen

Das GNU-Projekt hatte Anfang der 1990er Jahre zwar schon viele Programme der Anwendungsschicht geschrieben, einen Kernel gab es aber noch nicht.

Da kam das Linux-Projekt von Linus Torvalds, einem finnischen Studenten gerade recht: er veröffentlichte im Dezember 1992 den Linux-Kernel 0.99 unter der GNU GPL.

Damit war das erste freie Unix-System komplett und einsatzbereit.

25 Jahre später hat der Kernel die Version 4 und es arbeiten Hunderte an seiner Weiterentwicklung mit.

Da auch Android auf Linux basiert und es über eine Milliarde (!) Android-Geräte aber nur ca. 400 Millionen PCs gibt, ist Linux das mit Abstand häufigste Betriebssystem auf der Erde.

1.3.2 Open Source Initiative

Vielen Leuten war und ist die GNU GPL zu restriktiv und sie stören sich an der strengen, moralischen Auslegung des Begriffs *Freiheit* innerhalb der GNU GPL.

Aus diesem Grund haben 1998 Eric S. Raymond, Bruce Perens und Tim O'Reilly die *Open Source Initiative* gegründet.

Diese Gruppe verfolgt einen etwas pragmatischeren, marktfreundlichen Ansatz. Der Schwerpunkt liegt dabei hauptsächlich auf der freien *Zugänglichkeit* des Quellcodes, kommmerzielle Verwendung wird aber nicht unbedingt ausgeschlossen.

Die Idee war, durch etwas weniger strenge Open Source Lizenzen (s.u.) die Verbreitung von Open Source Software zu fördern, da in der geldgetriebenen Wirtschaft die GNU GPL nicht gerne gesehen ist.

Weitere Open Source Lizenzen

Allen folgenden Lizenzen ist gemein, dass der Quellcode veröffentlicht werden muss.

BSD-Lizenz Im Gegensatz zur GNU GPL dürfen Unternehmen Quellcode unter der BSD-Lizenz in eigene Produkte übernehmen und diese dann aber ohne Quellcode verkaufen. D.h. der Anwender einer Software unter der BSD-Lizenz kann mit ihr machen was er möchte, er darf sich nur nicht als ihr ursprünglicher Autor ausgeben.

Apache-Lizenz Die Apache-Lizenz funktioniert ähnlich wie die BSD-Lizenz: Software-Produkte dürfen ohne Quellen vermarktet werden. Die Apache-Lizenz ist juristisch etwas ausgefeilter als die BSD-Lizenz und regelt auch Fragen zu Patenten, Markenrechten, usw.

1.4 ICT Skills and Working in Linux

Weight: 2

ICT: Information and Communication Technology

Key Knowledge Areas:

- Desktop Skills
- Getting to the Command Line
- Industry uses of Linux, Cloud Computing and Virtualization

Terms and Utilities:

- Using a browser, privacy concerns, configuration options, searching the web and saving content
- Terminal and Console

- Password issues
- Privacy issues and tools
- Use of common open source applications in presentations and projects

2 Finding Your Way on a Linux System

2.1 Command Line Basics

Weight: 3
Key Knowledge Areas:

- Basic shell
- Command line syntax
- Variables
- Globbing
- Quoting

Terms and Utilities:

- Bash
- `echo`
- `history`
- PATH env variable
- `export`
- `type`

2.1.1 Bedienung der Kommandozeile

Obwohl heute über eine Milliarde *tragbarer Unterwegsfernsprecher* (Mobiltelefone) völlig ohne Tastatur daherkommen, lassen sich immer noch viele Aufgaben durch ein paar Texteingaben in ein Kommandozeilenprogramm weit schneller als mit einer grafischen Benutzeroberfläche erledigen.

Auf Linux-Systemen gibt es dazu ein spezielle Programme, die die Texteingaben eines Benutzers entgegennehmen, *interpretieren* und ausführen.

Im Schalenmodell des Betriebssystems stellen sie die Schicht zwischen Anwender und Programmen dar und man nennt sie aus diesem Grund *shell*-Programme.

Shell heisst auf deutsch Muschel oder Schale. Das soll verbildlichen, dass die Shell als weitere Schicht das Betriebssystem (also Kernel und Systemprogramme) umschliesst. Die Shellprogramme werden damit zu einer Anpassungsschicht zwischen dem Benutzer und dem Betriebssystem.

Die Abb. 2 zeigt dies am Beispiel des Programms `bash`. Das Programm `bash` ist unter Linux und auch anderen UNIX-Systemen (etwa Apple-OSX) absoluter Standard.

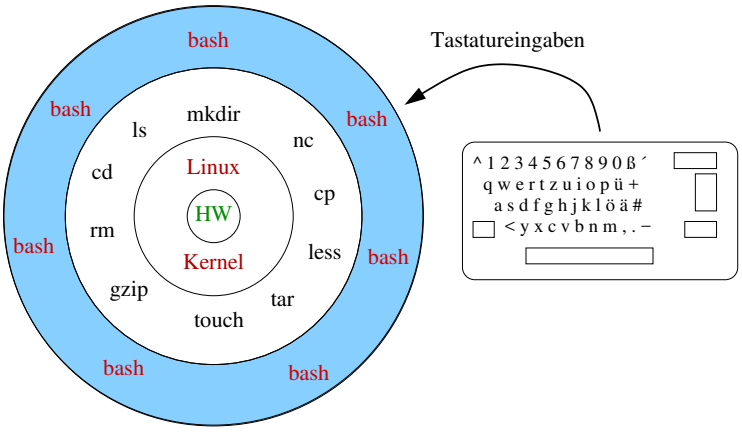

Abbildung 2: Schichtenmodell von Kernel und Anwendungsprogrammen und bash

> Hauptaufgabe der `bash` ist es, Befehle des Benutzers entgegenzunehmen und zu prüfen, ob die Eingabe mit einem Programm des Systems übereinstimmt und wenn ja, dieses Programm zu starten.

Auch unter Windows gibt es eine Shell, das Programm `cmd.exe`.

Die `bash` ist aber ungleich vielseitiger und mächtiger als `cmd.exe`. Ausserdem muss man beachten: `command` und `cmd` *ignorieren* Gross- und Kleinschreibung, **die `bash` unterscheidet hier aber sehr streng**: Das Kommando `cd` zum Wechseln eines Verzeichnisses kann man unter DOS als `CD, cD, Cd` oder `cd` eingeben, `bash` versteht aber nur die kleingeschriebene Version `cd`.

> Eingaben in der Shell (bash) werden strikt nach Gross- und Kleinschreibung unterschieden!

Ab Version 10 steht auch die bash auf Windows-Rechnern zur Verfügung. Bei Apple MacOSX ist die Bash in allen OSX-Versionen am Start.

Ein paar Fragen zur bash:

1. Starte einen Terminal-Prozess. Entweder über das Dock am linken Bildschirmrand, oder durch Drücken von ALT + F1 und dann Eingabe des Suchmusters **`term`**.
2. Was passiert, wenn man im Terminal CTRL + **+** bzw. CTRL + **-** drückt?
3. Wenn man wiederholt die ENTER-Taste drückt, erscheint am Zeilenanfang jedesmal ein Text. Wie wird dieser Text genannt?
4. Führt folgende Befehle aus und beobachtet die Änderung dieses Textes:

```
cd ~
cd /tmp
mkdir ./newDir
cd newDir
cd /
cd /etc
cd
```

5. Wie ist dieser Text aufgebaut? Gibt es spezielle Sonderzeichen? Gibt es spezielle Zeichenketten zwischen den Sonderzeichen? Welche Bedeutung haben diese Zeichenketten?

Konventionen beim Aufruf von Kommandos und bei der Eingabe von Parametern:

Die meisten Linux-Kommandos sind in C geschrieben. C wurde extra für die Entwicklung von UNIX entwickelt. Beim Aufruf eines Kommandos kann man Parameter übergeben. Mit folgendem C-Programm kann man testen, wie die Parameterübergabe an ein Programm funktioniert:

```
#include <stdio.h>

int main(int argc, int *argv[]){
  int i;
  for (i=0; i< argc; i++){
    printf("parameter Nr. %d: %s \n", i, argv[i]);
  }
}
```

2.1.2 Die 4 wichtigsten Tasten

Da die `bash` nun hauptsächlich mit der Tastatur bedient wird, muss man einigermassen gut tippen können, um flüssig arbeiten zu können.
Und damit die Tipperei noch schneller geht, gibt es einige wichtige Tasten, die einem viel Tipparbeit ersparen:

1. ENTER

   Die allerwichtigste Taste ist natürlich die ENTER-Taste.

2. CURSOR-HOCH

   Wenn man mit ENTER ein Kommando abgeschickt hat und nun feststellt, dass irgendwo ein Fehler ist, muss man die Zeile nicht nochmals tippen: mit der Taste CURSOR-HOCH kann man den letzten Befehl wiederholen. Bei mehrmaligem Betätigen von CURSOR-HOCH erreicht man alle Kommandos, die man in der Vergangenheit abgeschickt hatte. Wie viele Kommandozeilen von der `bash` gespeichert werden, lässt sich konfigurieren (Datei `~/.bashrc`).

3. CURSOR-RUNTER

   Mit der Taste CURSOR-RUNTER kann man in der Kommandozeilen-Geschichte wieder Richtung Gegenwart blättern.

4. TAB

   Die TAB -Taste sorgt dafür, dass die Arbeit mit der `bash` überhaupt erst Spass machen kann: Die TAB -Taste *expandiert* Programm-, Datei- und Verzeichnisnamen. Und mit expandieren ist *vervollständigen* gemeint: Drückt man TAB , versucht die `bash` den jeweils benötigten Namen automatisch zu vervollständigen. Welcher Name vervollständigt wird, hängt davon ab, an welcher Stelle einer Kommandozeile TAB betätigt wird. Im nächsten Unterkapitel (Kap. 2.1.3) steht dazu noch Näheres.

2.1.3 Kommandozeilen und Kommandoexpansion

Eine Kommandozeile ist ein Programmname gefolgt von einem oder mehreren Parametern. Allgemein sieht das so aus:

kommandoname ␣ *optionen* ␣ *pfad*

Dabei steht *pfad* für eine Datei oder ein Verzeichnis, ggfs. mit einer relativen oder absoluten Pfadangabe, wo sich die Datei oder das Verzeichnis befinden.

Die Optionen müssen nicht immer vorhanden sein und es gibt sogar Kommandos, die ohne Parameter aufgerufen werden (z.B. `ps`: processes).

Ganz wichtig sind die Leerzeichen: ␣ . Sie dienen der `bash` als Trennzeichen. Z.B. wechselt die `bash` nach der Eingabe von `cd` ␣ `..` ins darüberliegende Verzeichnis. Quittiert aber die Eingabe von `cd..` mit

```
bash: cd..: command not found
```

Das heisst, die `bash` sucht nach einem Kommando, das `cd..` heisst (eben ”cd..” als Ganzes) und dieses Kommando gibt es normalerweise nicht.

Die Wirkung der TAB -Taste ist nun innerhalb einer Zeile so:

- Ist man gerade bei der Eingabe des Kommandonamens und drückt die TAB -Taste, vervollständigt die `bash` den Kommandonamen, falls dieser schon eindeutig ist.

  Beispiel: nach Eingabe von `fire` TAB drücken und die `bash` vervollständigt zu `firefox`.

  Drückt man bereits nach der Buchstabenfolge `fi` auf TAB , piepst der Rechner zur Warnung, dass die Eingabe noch nicht eindeutig ist. Drückt man dann nochmals TAB , werden alle Kommandos gelistet, die mit `fi` beginnen. Dabei sucht die `bash` alle Verzeichnisse ab, die im Suchpfad (`$PATH`) stehen. Nun kann man durch Eingabe von weiteren Buchstaben die Eingabe genauer machen, bis man das gewünschte Kommando dastehen hat oder TAB es vollständig expandiert.

- Ist man bei der Eingabe eines Verzeichnis- oder Dateinamens innerhalb einer Kommandozeile, versucht die `bash` diese ebenfalls zu vervollständigen. Das funktioniert wie beim Kommandonamen, nur sucht die `bash` nun nach *Datei- oder Verzeichnisnamen*.

2.2 Using the Command Line to Get Help

Weight: 2
Key Knowledge Areas:

- Man
- Info

Terms and Utilities:

- `man`
- `info`
- Man pages
- `/usr/share/doc/`
- `locate`

2.3 Using Directories and Listing Files

Weight: 2
Key Knowledge Areas:

- Files, directories
- Hidden files and directories
- Home
- Absolute and relative paths

Terms and Utilities:

- Common options for ls
- Recursive listings
- `cd`
- . and ..
- `home` and ~

2.3.1 Files and Directories - Dateien und Verzeichnisse

Auf UNIX-Systemen, also auch unter Linux gilt:

Alles ist eine Datei!

D.h. Geräte (z.B. Tastatur, Bildschirm, Festplatten), Netzwerk- und USB-schnittstellen und vieles mehr werden vom Kernel wie eine Datei behandelt.

Auch *Verzeichnisse* selbst sind Dateien!

Das aktuell am meisten verbreitete Dateisystem unter Linux heisst **ext4**. Es funktioniert aber ähnlich wie alle Dateisystem unter Unix.

Da ext4 etwas kompliziert geworden ist, schauen wir uns hier exemplarisch für alle Unix-Dateisysteme die Details seines Vorgängers **ext2** an:

- Die Magnetspuren auf Festplatten werden von der Hardware in **Segmente** von 512 Bytes Länge unterteilt. Dies ist die kleinste Einheit, die auf den Platten adressiert werden kann. Manchmal werden diese Segmente auch als Blöcke gekennzeichnet, im Unix-Umfeld ist 'Segment' jedoch die bessere Bezeichnung (s.u.).
- Je 8 HD-Segmente werden im Dateisystem zu **Blöcken** von 4KiBytes ($4{\cdot}2^{10} Bytes$) zusammengefasst. Blockgrössen von 2KiB oder 1KiB sind auch möglich. In Windows-Dateisystemen, nennt man die Zusammenfassung von Segmenten Cluster.
- Die nächste und wichtigste Organisationseinheit von Unix-Dateisystemen sind die sog. *Inodes*. Die Inodes sind winzige Dateien von einheitlich 128 Bytes Länge, die **ausser dem Dateinamen** (!!) alle Attribute der eigentlichen Datendatei enthalten. Die Attribute werden *Meta-Daten* genannt. Das wären:
  - Ein Zähler, der die Anzahl der Verweise (=Dateinamen in Verzeichnissen) auf diesen Inode enthält. Fachausdruck Hard-Link-Zähler.
  - Besitzer und Gruppe
  - Zugriffsrechte
  - Typ der Datei: Datei, Verzeichnis, Verknüpfung (Link),
  - Grösse der Datei
  - Diverse Zeitstempel
  - Eine Liste mit den Nummern von bis zu 12 Blöcken (bei alten Unix-Dateisystemen: 10 Blöcke), die die eigentlichen Dateidaten enthalten. Anders ausgedrückt: der Inode enthält 12 Zeiger auf die Datenblöcke, die die eigentlichen Dateidaten enthalten.
  - Bei grossen Dateien enthält der 13. Listenplatz nicht die Blocknummer (Zeiger) direkt, sondern die Nummer eines Blocks, der wiederum eine Liste mit bis zu 256 Blocknummern enthält (**einfach indirekter Block**).
  - Wird noch mehr Platz benötigt, steht an 14. Stelle ein **zweifach indirekter Block** und evtl. an 15. Stelle **dreifach indirekter Block**.
- Jeder Inode ist mit einer eindeutigen Schlüsselnummer (**Inode-Nummer**) markiert. Diese wird nicht im Inode gespeichert, sondern ergibt sich aus dem physikalischen Speicherort des Inodes selbst: Alle Inodes haben die einheitliche Länge von 128 Bytes und werden hintereinander weg als Liste geschrieben. Diese Liste enthält im Normalfall 2048 Inodes.

Über die Inode-Nummer ist der eindeutige Zugriff auf eine bestimmte Datei möglich.

- Verzeichnisse werden unter Unix auch als Inodes und damit als Mini-Dateien (s.o.) verwaltet. Ein Verzeichnis ist demnach eine Datei, die einfach nur die Namen der enthaltenen Dateien und Unterverzeichnisse enthält und diese mit den zugehörigen Inode-Nummern verknüpft.

  Durch diese Verknüfpungen ist es möglich, einer physikalischen Datei mehrere Namen zu geben (Siehe Übungsaufgabe **??**).

> Die Verknüpfung einer Inode-Nummer mit einem Dateinamen nennt man **Hardlink**. Zu einem Inode darf es beliebig viele Dateinamen (Hardlinks) geben.

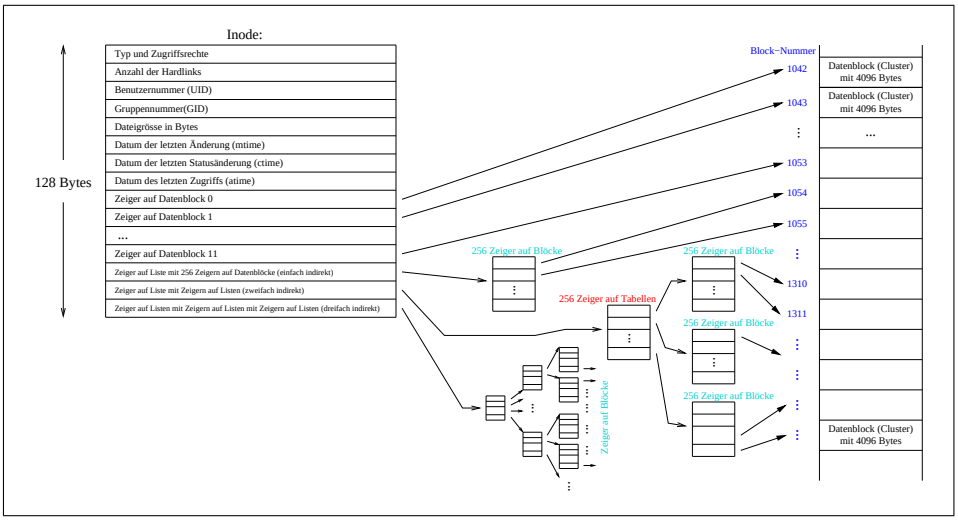

Abbildung 3: Modell eines Linux-Inodes

Hier ein Beispiel für den Inhalt eines Inodes, der ein Verzeichnis repräsentiert. Erzeugt wird solch eine Ausgabe mit dem Kommando

```
ls -lia
```

```
 22:51:34|micha@pellesUbu:~/schule/LaTeXDocs/lpicEssentials$ ls -lia
insgesamt 408
32768400 drwxr-xr-x  3 micha micha   4096 Sep 25 22:51 .
28051306 drwxr-xr-x 69 micha micha   4096 Sep 13 22:41 ..
35259246 drwxr-xr-x  2 micha micha   4096 Sep 25 22:41 bilder
32770404 -rw-rw-r--  1 micha micha   4482 Sep 25 22:41 lpicEssentials.aux
32768431 -rw-rw-r--  1 micha micha  37915 Sep 25 22:41 lpicEssentials.log
32770408 -rw-rw-r--  1 micha micha 240973 Sep 25 22:41 lpicEssentials.pdf
32770413 -rw-r--r--  1 micha micha  32598 Sep 25 22:45 lpicEssentials.tex
32770409 -rw-r--r--  1 micha micha  24539 Sep 21 11:32 lpicEssentials.tex~
32770406 -rw-rw-r--  1 micha micha   3116 Sep 25 22:41 lpicEssentials.toc
32770424 -rw-r--r--  1 micha micha   4251 Apr 28  2013 newunicodechar.sty
32770410 -rw-r--r--  1 micha micha   5814 Sep 21 00:43 schichten.tex
```

Besonders wichtig für das Verständnis des Dateisystems ist der Eintrag mit dem Dateinamen **..**, das *übergeordnete Verzeichnis*.

Im Beispiel hat dieser die Inode-Nr. 32768400.

Jedes Verzeichnis hat auch einen Eintrag mit dem Namen **.** . Dieser einzelne Punkt steht für das Verzeichnis selbst, in dem man sich befindet.

Verzeichnisse in Linux-Dateisystemen sind also nichts anderes als Tabellen, bestehend aus Inode-Nr. und Dateinamen.

Da diese Tabellen immer auch die Inode-Nummer der übergeordneten Tabelle (=Verzeichnis) enthalten, entsteht eine *Baumstruktur*.

Beim allerobersten Verzeichnis, haben der Eintrag **.** und **..** die selbe Inode-Nr. , d.h. dieses Verzeichnis hat keinen Vorgänger mehr. Man nennt es das *Wurzelverzeichnis* (root-Directory).

Übungsaufgaben

Gegeben sind die Inhalte einiger Inodes:

Inode-Nr.	Name
2	.
2	..
169	A

Inode-Nr.	Name
169	.
2	..
170	B

Inode-Nr.	Name
170	.
169	..
171	C1
173	C2

Inode-Nr.	Name
171	.
170	..
172	D1

Inode-Nr.	Name
173	.
170	..
174	D2

Inode-Nr.	Name
172	.
171	..
175	dummy

Inode-Nr.	Name
174	.
173	..

Skizzieren Sie zunächst die entsprechende Verzeichnisstruktur als Baum und erzeugen Sie sie anschliessend mit den Befehlen **`mkdir`** und **`cd`** eine solche Struktur in Ihrem Heimatvereichnis. Natürlich hat Ihr Heimatverzeichnis ein übergeordnetes Verzeichnis, die Inodes von **.** und **..** Ihres Heimatverzeichnisses sind also verschieden.

Mir `mv` *alterName neuerName* können Sie ein Verzeichnis umbenennen, wenn Sie sich vertippt haben.

Verwenden Sie bitte noch nicht den Befehl `rm -rf`

In einem zweiten Versuch sollen Sie zunächst folgende Kommandos abschicken:

```
cd /tmp
mkdir -p a/b/c/d
mkdir -p a/b/e/f
cd a/b/c/d
touch testdatei
ls -i #inode-nummer merken!
cd /tmp/a/b/e/f
ln /tmp/a/b/c/d/testdatei harterlink
ls -i #inode-nummer merken!
cd /tmp
tree --inodes a
```

Fragen:

- Was bewirkt die Option -p beim Kommando `mkdir` ?
- Was macht das Kommando `touch`, wenn man es ohne Parameter einfach so eingibt?
- Was macht das Kommando `ln` ?

2.3.2 Hidden files and directories

Versteckte Dateien oder Verzeichnisse sind nicht wirklich versteckt, sondern ihr Name beginnt einfach nur mit einem Punkt. Beispiele:

```
.bashrc
.ssh
..
.config
```

Übungsaufgaben

Testen Sie folgende Befehle:

```
cd ~
ls
ls -a
less .bash_history
```

Was bewirkt die Option -a beim Kommando *ls* ?

2.3.3 Home

Dateien und Verzeichnisse (die ja auch Dateien sind) haben unter allen modernen Betriebssystemen einen *Eigentümer*.

Benutzer, die ein Benutzerkonto auf einem Linux-Rechner haben, besitzen dort *automatisch* einen ganzen *Ast* des Dateibaums. Dieser Ast wird *Heimatverzeichnis* engl. *home directory* oder kurz *home* genannt.

Dieser Ast liegt i.d.R im Verzeichnis `/home`. Hier in der Schule ist `/home` nochmals in Äste für Lehrer, Schüler und Klassen unterteilt.

Damit man nun nicht lange suchen und tippen muss, wenn man nach Hause möchte, gibt es ein spezielles Zeichen, das als Abkürzung für den Pfadnamen ins eigene Heimatverzeichnis steht: **~**

Übungsaufgaben

Testen Sie folgende Befehle:

```
cd /etc/apache2
pwd
ls ~
cd ~
pwd
```

Was bewirkt das Kommando `pwd`? Für welchen Pfad steht bei Ihnen die Tilde (~)?

2.3.4 Absolute and relative paths

- Ein absoluter Pfad beginnt immer mit dem Zeichen "/". Bei einer solchen Pfadangabe sucht das System **beim *Wurzelverzeichnis* beginnend den angegebenen Pfad ab. Diese Suche ist unabängig von aktuellen Verzeichnis.**
- Ein relativer Pfad beginnt mit einem Datei- oder Verzeichnisnamen. Der Verzeichnisname kann natürlich auch mit `./` oder `../` beginnen.

  Ein relativer Pfad wird vom *aktuellen* Verzeichnis ausgehend abgesucht!

Testen Sie folgende Befehle und skizzieren Sie dabei auf einem Blatt die Verzeichnsistruktur die angelegt wird und die Pfade, die Sie dort im Baum angeben:

```
cd ~
mkdir -p /tmp/x/y/z
mkdir -p /tmp/x/y/u/v
cd /tmp/x/y/z
touch ./datInZ
touch ../datInY
touch ../../datInX
touch ../u/v/datInV
tree /tmp/x
cd ~
rm /tmp/x/y/u/v/datInV
```

2.4 Creating, Moving and Deleting Files

Weight: 2
Key Knowledge Areas:

- Files and directories

- Case sensitivity
- Simple globbing and quoting

Terms and Utilities:

- `mv, cp, rm, touch`
- `mkdir, rmdir`

Die grundlegenden Aufgaben im Dateisystem lassen sich mit folgenden 6 Kommandos erledigen:

2. **list**

   ```
   ls [-optionen] [pfad]
   ```

   Damit wird der Inhalt des aktuellen Verzeichnisses gelistet, wenn keine Pfadangabe gemacht wurde, ansonsten wird eben das mit der Pfadangabe bezeichnete Verzeichnis aufgelistet.

   Beispiele:

   `ls`: erzeugt eine Liste der sichtbaren Dateien. Die Ausgabe erfolgt platzsparend mit mehreren Dateinamen pro Ausgabezeile.

   `ls -l`: die Option `-l` erzeugt eine ausführlichere Liste. Pro Zeile wird nur eine Datei aufgelistet, dafür werden aber wichtige Dateiattribute wie Ausführrechte, Besitzer, Gruppe, Dateigrösse, Datum und Uhrzeit ausgegeben.

   `ls -la`: die zusätzliche Option a steht für **all**. D.h. es werden jetzt auch versteckte Dateien aufgelistet. Versteckte Dateien sind alle Dateien, die mit *einem Punkt* beginnen.

   `ls -la > listing`: bei diesem Befehl wird die Ausgabe, die ja normalerweise auf dem Bildschirm landet, in eine Datei *umgeleitet*. Das wird durch das Zeichen '>' erreicht. Diese Umleitung funktioniert mit jedem Programm, das auf dem Bildschirm ausgibt.

3. **change directory**

   ```
   cd <pfad>
   ```

   Damit wechselt man ins angegebene Verzeichnis. Die Pfadangabe kann absolut, also vom Wurzelverzeichnis aus, oder relativ, vom aktuellen Verzeichnis ausgehend, angegeben werden.

4. **move**

   ```
   mv [-opt] <name_alt> <name_neu>
   mv [-opt] <dateien> <verzeichnis>
   ```

   mv dient zum Verschieben und Umbenennen von Dateien. Eine Datei wird umbenannt, wenn man sie innerhalb des selben Verzeichnisses verschiebt.

5. **copy**

```
cp [-opt] <quelle> <ziel>
cp [-opt] <dateien> <verzeichnis>
```

cp kopiert Dateien.

Optionen:

`-R`: kopiere rekursiv. Mit dieser Option lassen sich Verzeichnisse, ja ganze Pfade kopieren.

`-u`: ist diese Option gesetzt wird nur kopiert, wenn keine neuere Datei gleichen Namens überschrieben wird.

6. **make directory**

```
mkdir [-opt] <name>
```

Mit `mkdir` wird ein neues Verzeichnis angelegt.

Optionen:

`-p`: mit dieser Option lassen sich Verzeichnisse und Unterverzeichnisse auf einmal anlegen.

Beispiel:

`mkdir -p ~/linuxUebung/loesung` legt zuerst das Verzeichnis `linuxUebung` und innerhalb dieses Verzeichnisses das Unterverzeichnis `loesung` an.

7. **remove**

```
rm [-opt] <name>
```

Mit `rm` lassen sich Dateien und bei Verwendung von `-r` ganze Pfade löschen. Vorsicht: `rm` fragt nicht zurück!

3 The Power of the Command Line

3.1 Archiving Files on the Command Line

Weight: 2
Key Knowledge Areas:

- Files, directories
- Archives, compression

Terms and Utilities:

- `tar`
- Common tar options
- `gzip, bzip2`
- `zip, unzip`

8. **tape archive**

   `tar <aktion>f <archivdatei> [-C] <pfad>`

   `tar` dient dazu, Archivdateien zu erzeugen, oder Archivdateien wieder auszupacken. Ob nun gepackt oder entpackt werden soll, wird mit `<aktion>` gesteuert: **x** steht für *extrakt*, also auspacken und **c** steht für *create*, einpacken.

   Mit der Option **z** wird bei diesen Vorgängen entsprechend noch komprimiert oder dekomprimiert (verwendet wird dazu gzip= GNU-Zip). Die Option **v** (verbose) schaltet die Ausgabe der internen Abläufe auf dem Bildschirm ein.

   Die wichtigste Option ist aber das `f`: ursprünglich war `tar` dazu gedacht, Archive auf einem Bandlaufwerk zu schreiben oder lesen. Damit `tar` stattdessen auf eine Archiv**datei** auf der Platte zugreift, muss man immer die Option `f` verwenden, auf die der entsprechende Dateiname folgen musss.

   Beispiele:

   `tar -cvzf archiv.tgz .`: Archiviert und komprimiert das aktuelle Verzeichnis (.) in der Datei `archiv.tgz`

   `tar -cvzf briefe.tgz briefe`: Archiviert und komprimiert das Verzeichnis `briefe` in der Datei `archiv.tgz`

   `tar -xvzf archiv.tgz`: Extrahiert und entkomprimiert das Archiv ins aktuelle Verzeichnis.

   `tar -xvzf archiv.tgz -C ~/dummy`: Extrahiert und entkomprimiert das Archiv ins Verzeichnis `/dummy`.

3.2 Searching and Extracting Data from Files

Weight: 3
Key Knowledge Areas:

- Command line pipes
- I/O re-direction
- Basic Regular Expressions ., [ ], *, ?

Terms and Utilities:

- `grep`
- `less`
- `cat, head, tail`
- `sort`
- `cut`
- `wc`

3.3 Turning Commands into a Script

Weight: 4
Key Knowledge Areas:

- Basic shell scripting
- Awareness of common text editors

Terms and Utilities:

- `#! (shebang)`
- `/bin/bash`
- Variables
- Arguments
- for loops
- `echo`
- Exit status

4 The Linux Operating System

4.1 Choosing an Operating System

Weight: 1
Key Knowledge Areas:

- Windows, Mac, Linux differences
- Distribution life cycle management

Terms and Utilities:

- GUI versus command line, desktop configuration
- Maintenance cycles, Beta and Stable

4.2 Understanding Computer Hardware

Weight: 2
Key Knowledge Areas:

- Hardware

Terms and Utilities:

- Motherboards, processors, power supplies, optical drives, peripherals
- Hard drives and partitions, `/dev/sd*`
- Drivers

4.3 Where Data is Stored

Weight: 3
Key Knowledge Areas:

- Programs and configuration, packages and package databases
- Processes, memory addresses, system messaging and logging

Terms and Utilities:

- `ps, top, free`
- `syslog, dmesg`
- `/etc/, /var/log/`
- `/boot/, /proc/, /dev/, /sys/`

4.4 Your Computer on the Network

Weight: 2
Key Knowledge Areas:

- Internet, network, routers
- Querying DNS client configuration
- Querying Network configuration

Terms and Utilities:

- `route, ip route show`
- `ifconfig, ip addr show`
- `netstat, ip route show`
- `/etc/resolv.conf, /etc/hosts`
- IPv4, IPv6
- `ping`
- `host`

5 Security and File Permissions

5.1 Basic Security and Identifying User Types

Weight: 2
Key Knowledge Areas:

- Root and Standard Users
- System users

Terms and Utilities:

- `/etc/passwd, /etc/group`
- `id, who, w`
- `sudo, su`

5.2 Creating Users and Groups

Weight:
Key Knowledge Areas:

- User and group commands
- User IDs

Terms and Utilities:

- `/etc/passwd, /etc/shadow, /etc/group, /etc/skel/`
- `id, last`
- `useradd, groupadd`
- `passwd`

5.3 Managing File Permissions and Ownership

Weight: 2
Key Knowledge Areas:

- File/directory permissions and owners

Terms and Utilities:

- `ls -l, ls -a`
- `chmod, chown`

5.4 Special Directories and Files

Weight: 1
Key Knowledge Areas:

- Using temporary files and directories
- Symbolic links

Terms and Utilities:

- `/tmp/`, `/var/tmp/` and Sticky Bit
- `ls -d`
- `ln -s`