Linux
Professional
Institute

Linux Essentials

Version 1.6
Deutsch

010

Table of Contents

Linux
Professional
Institute

Thema 1: Die Linux-Community und Karriere im Open-Source-Umfeld

1.1 Die Entwicklung von Linux und gängige Betriebssysteme

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 1.1

Gewichtung

2

Hauptwissensgebiete

- Distributionen
- Embedded Systems
- Linux in der Cloud

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- Debian, Ubuntu (LTS)
- CentOS, openSUSE, Red Hat, SUSE
- Linux Mint, Scientific Linux
- Raspberry Pi, Raspbian
- Android

1.1 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	1 Die Linux-Community und Karriere im Open-Source-Umfeld
Lernziel:	1.1 Die Entwicklung von Linux und gängige Betriebssysteme
Lektion:	1 von 1

Einführung

Linux ist eines der beliebtesten Betriebssysteme. Linus Torvalds begann mit der Entwicklung 1991. Es ist von Unix inspiriert, einem weiteren Betriebssystem, das in den 1970er Jahren von AT&T Laboratories entwickelt wurde. Unix war auf Kleincomputer ausgerichtet: Damals galten “kleine” Computer als Maschinen, die keine ganze Halle mit Klimaanlage benötigten und weniger als eine Million Dollar kosteten. Später verstand man darunter Maschinen, die von zwei Personen angehoben werden konnten. Zu dieser Zeit war ein erschwingliches Unix-System auf Rechnern wie Bürocomputern, die in der Regel auf der x86-Plattform basierten, nicht ohne weiteres verfügbar. So begann Linus, damals noch Student, mit der Implementierung eines Unix-ähnlichen Betriebssystems für diese Plattform.

Meist folgt Linux denselben Grundideen wie Unix, aber Linux selbst enthält keinen Unix-Code — es ist ein unabhängiges Projekt. Linux wird nicht von einem einzelnen Unternehmen, sondern von einer internationalen Gemeinschaft von Programmierern getragen, ist frei verfügbar und von jedermann uneingeschränkt zu nutzen.

Distributionen

Eine *Linux-Distribution* ist eine Sammlung, die aus einem Linux-Kern (dem *Kernel*) und einer Auswahl von Anwendungen besteht und die von einem Unternehmen oder einer Benutzergemeinschaft gepflegt wird. Ziel ist es, den Kernel und die Anwendungen, die auf dem Betriebssystem laufen, für einen bestimmten Anwendungsfall oder eine bestimmte Benutzergruppe zu optimieren, weshalb einige Distributionen oft spezifische Werkzeuge für die Softwareinstallation und Systemadministration umfassen. So werden einige Distributionen hauptsächlich für Desktop-Umgebungen verwendet, die einfach zu bedienen sein müssen, während andere vor allem auf Servern laufen, um die verfügbaren Ressourcen so effizient wie möglich zu nutzen.

Eine weitere Möglichkeit, Distributionen zu klassifizieren, besteht in der Zuordnung zu einer *Distributionsfamilie*: Distributionen der Debian-Familie nutzen den Paketmanager `dpkg`, um die auf dem Betriebssystem ausgeführte Software zu verwalten. Pakete, die dieser Paketmanager verwaltet, werden von Mitgliedern der Community der Distribution gepflegt. Diese *Maintainer* verwenden das Paketformat `deb` und legen damit fest, wie die Software auf dem Betriebssystem installiert wird und wie sie standardmäßig konfiguriert ist. Genau wie eine Distribution ist ein Paket eine Software-Sammlung mit entsprechender Konfiguration und Dokumentation, die es dem Benutzer erleichtert, die Software zu installieren, zu aktualisieren und zu nutzen.

Debian GNU/Linux ist die größte Distribution der Debian-Distributionsfamilie. 1993 startete Ian Murdock das Debian GNU/Linux-Projekt, das heute Tausende von Freiwilligen unterstützen. Ziel von Debian GNU/Linux ist ein sehr zuverlässiges Betriebssystem. Zugleich steht es für Richard Stallmans Vision eines Betriebssystems, das die Freiheiten des Benutzers zum Ausführen, Studieren, Verteilen und Verbessern der Software respektiert.

Ubuntu ist eine weitere erwähnenswerte Debian-basierte Distribution. 2004 wurde Ubuntu von Mark Shuttleworth und seinem Team gegründet, mit dem Ziel, eine einfach zu bedienende Linux-Desktop-Umgebung zu schaffen. Ubuntu hat die Mission, auf der ganzen Welt freie Software bereitzustellen und die Kosten für professionelle Dienstleistungen zu senken. Ein neues Release der Distribution erscheint planmäßig alle sechs Monate und in einer Version mit längerfristigem Support alle zwei Jahre.

Red Hat ist eine Linux-Distribution, die von dem gleichnamigen Softwareunternehmen entwickelt und gepflegt wird, das 2019 von IBM übernommen wurde. 1994 wurde die Red Hat Linux-Distribution gestartet und 2003 in *Red Hat Enterprise Linux* (RHEL) umbenannt. Es dient Unternehmen als zuverlässige Enterprise-Lösung, die von Red Hat unterstützt wird und Software umfasst, die den Einsatz von Linux in professionellen Serverumgebungen erleichtert. Einige Komponenten erfordern kostenpflichtige Subskriptionen oder Lizenzen. Das *CentOS*-Projekt nutzt den frei verfügbaren Quellcode von Red Hat Enterprise Linux und fügt ihn zu einer Distribution

zusammen, die kostenlos bereitsteht, dafür aber keinen kommerziellen Support bietet.

Sowohl RHEL als auch CentOS sind für den Einsatz in Serverumgebungen optimiert. Das Projekt *Fedora* wurde 2003 gegründet und erstellt eine Linux-Distribution für Desktop-Computer. Red Hat hat die Fedora-Distribution initiiert und pflegt sie seitdem. Fedora ist sehr fortschrittlich und übernimmt neue Technologien sehr schnell; sie gilt darum oft als Test für neue Technologien, die später möglicherweise in RHEL aufgenommen werden. Alle Red Hat-basierten Distributionen verwenden das Paketformat `rpm`.

Das Unternehmen SUSE wurde 1992 in Deutschland als Unix-Service-Provider gegründet. 1994 veröffentlichte es die erste Version von *SUSE Linux*, bekannt vor allem für sein Konfigurationstool YaST, mit dem Administratoren Soft- und Hardware installieren und konfigurieren sowie Server und Netzwerke einrichten. Ähnlich wie bei RHEL gibt es bei SUSE den *SUSE Linux Enterprise Server* als kommerzielle Variante. Mit selteneren Releases wendet es sich vor allem an Unternehmen. Es wird sowohl als Server- wie auch als Desktop-Umgebung mit entsprechenden Paketen angeboten. 2004 startete SUSE das Projekt *openSUSE*, mit dem Entwickler und Anwender künftige Technologien für das System testen und weiterentwickeln können. Die openSUSE-Distribution steht zum kostenlosen Download zur Verfügung.

Im Laufe der Jahre entstanden weitere unabhängige Distributionen. Einige basieren auf Red Hat oder Ubuntu, andere wollen bestimmte System- oder Hardwareeigenschaften verbessern. Es gibt Distributionen mit ganz spezifischen Funktionalitäten wie *QubesOS*, eine sehr sichere Desktop-Umgebung, oder *Kali Linux* zur Ermittlung von Software-Schwachstellen, das insbesondere Penetration-Tester nutzen. Zuletzt entstanden zahlreiche sehr kleine Linux-Distributionen für den Betrieb in Linux-Containern wie Docker. Es gibt auch Distributionen, die speziell für Komponenten von Embedded Systemen und sogar Smart Devices entwickelt wurden.

Embedded Systeme

Embedded Systeme sind eine Kombination aus Computer-Hardware und -Software mit sehr spezifischen Aufgaben innerhalb eines größeren Systems. In der Regel sind sie Teil anderer Geräte und dienen deren Steuerung. Embedded Systeme finden sich in Automobil-, Medizin- und sogar Militäranwendungen. Aufgrund seiner vielfältigen Einsatzmöglichkeiten entstanden zahlreiche Betriebssysteme auf Basis des Linux-Kernels für den Einsatz in Embedded Systemen. Ein großer Teil der Smart Devices verfügt über ein Betriebssystem auf Basis des Linux-Kerns.

Daher sind Embedded Systeme auch mit Embedded Software verbunden. Zweck dieser Software ist es, auf die Hardware zuzugreifen und sie nutzbar zu machen. Die Hauptvorteile von Linux gegenüber anderer, proprietärer Embedded Software sind die Kompatibilität mit herstellerübergreifenden Plattformen, Entwicklung, Support und fehlende Lizenzgebühren. Zwei der beliebtesten Embedded Softwareprojekte sind *Android*, das hauptsächlich auf Mobiltelefonen

verschiedener Hersteller zum Einsatz kommt, und *Raspbian*, das vor allem auf dem Raspberry Pi verwendet wird.

Android

Android ist ein von Google entwickeltes mobiles Betriebssystem. 2003 wurde Android Inc. in Palo Alto, Kalifornien, gegründet. Das Unternehmen schuf zunächst ein Betriebssystem für Digitalkameras. 2005 kaufte Google Android Inc. und entwickelte eines der größten mobilen Betriebssysteme.

Die Basis von Android ist eine modifizierte Version des Linux-Kernels mit zusätzlicher Open-Source-Software. Das Betriebssystem ist hauptsächlich für Touchscreen-Geräte gedacht, aber Google hat auch Versionen für TV und Armbanduhren entwickelt. Weitere Varianten von Android gibt es für Spielekonsolen, Digitalkameras sowie PCs.

Android ist im Rahmen des *Android Open Source Project* (AOSP) frei verfügbar. Google bietet neben dem Open-Source-Kern von Android eine Reihe proprietärer Komponenten an, darunter Anwendungen wie Google Calendar, Google Maps, Google Mail, den Browser Chrome sowie den Google Play Store, der die einfache Installation von Apps ermöglicht. Die meisten Nutzer betrachten diese Tools als integrale Bestandteile von Android, und fast alle mobilen Geräte, die mit Android in Europa und Amerika ausgeliefert werden, enthalten proprietäre Google-Software.

Android auf Embedded Geräten hat viele Vorteile: Das Betriebssystem ist mit einer grafischen Benutzeroberfläche intuitiv zu bedienen, und es hat eine sehr große Entwicklergemeinschaft, so dass man leicht Hilfe bei der Entwicklung findet. Zudem wird es von der Mehrheit der Hardwareanbieter mit Android-Treibern unterstützt, so dass sich Prototypen von Systemen einfach und kostengünstig entwickeln lassen.

Raspbian und der Raspberry Pi

Raspberry Pi ist ein preiswerter, scheckkartengroßer Computer, der als vollwertiger Desktop-Rechner, aber auch innerhalb eines Embedded-Linux-Systems fungieren kann. Entwickelt wird er von der *Raspberry Pi Foundation*, einer gemeinnützigen Organisation mit Sitz in Großbritannien, die vor allem jungen Menschen das Programmieren und die Funktionsweise von Computern nahebringt. Der Raspberry Pi lässt sich so einrichten und programmieren, dass er die gewünschten Aufgaben oder Operationen innerhalb eines deutlich komplexeren Systems erfüllt.

Zu den Besonderheiten des Raspberry Pi gehört ein Satz von *General Purpose Input-Output* (GPIO)-Pins, über die man elektronische Geräte und Erweiterungsboards anschließt, so dass der Raspberry Pi zur Plattform für Hardwareentwicklung wird. Obwohl ursprünglich für Bildungszwecke gedacht, werden Raspberry Pis heute in verschiedenen DIY-Projekten sowie für

Industrial Prototyping bei der Entwicklung von Embedded Systemen eingesetzt.

Der Raspberry Pi verwendet ARM-Prozessoren. Verschiedene Betriebssysteme, darunter Linux, laufen auf dem Raspberry Pi. Da er keine Festplatte hat, startet das Betriebssystem von einer SD-Speicherkarte. Eine der bekanntesten Linux-Distributionen für den Raspberry Pi ist *Raspbian*. Wie der Name schon verrät, gehört es zur Debian-Distributionsfamilie, ist für die Installation auf der Raspberry Pi-Hardware angepasst und bietet mehr als 35000 für diese Umgebung optimierte Pakete. Neben Raspbian gibt es für den Raspberry Pi zahlreiche andere Linux-Distributionen, wie z.B. *Kodi*, das den Raspberry Pi in ein Media Center verwandelt.

Linux und die Cloud

Der Begriff *Cloud Computing* beschreibt eine standardisierte Methode zur Nutzung von Rechenleistung, entweder durch Einkauf bei einem Public-Cloud-Anbieter oder durch den Betrieb einer privaten Cloud. Studien besagen, dass 2017 über 90% der Public-Cloud-Arbeitslast von Linux-Systemen verarbeitet wurden. Jeder Cloud-Anbieter, von *Amazon Web Services* (AWS) bis *Google Cloud Platform* (GCP), bietet verschiedene Linux-Varianten an. Sogar Microsoft hat heute in seiner *Azure* Cloud Linux-basierte virtuelle Maschinen im Angebot.

Linux wird häufig im Rahmen von *Infrastructure as a Service* (IaaS) angeboten. IaaS-Instanzen sind virtuelle Maschinen, die innerhalb weniger Minuten in der Cloud bereitgestellt werden. Beim Start einer IaaS-Instanz wird ein Image ausgewählt, das in der neuen Instanz sämtliche Daten bereitstellt. Cloud-Provider bieten verschiedene Images an — sowohl betriebsbereite Installationen gängiger Linux-Distributionen als auch eigene Linux-Versionen. Der Cloud-Benutzer wählt ein Image mit seiner bevorzugten Distribution aus und kann kurz darauf auf eine Cloud-Instanz mit dieser Distribution zugreifen. Die meisten Cloud-Anbieter fügen ihren Images Tools hinzu, um die Installation an eine bestimmte Cloud-Instanz anzupassen. Diese Tools können beispielsweise die Dateisysteme des Images auf die aktuelle Festplatte der virtuellen Maschine erweitern.

Geführte Übungen

1. Inwiefern unterscheidet sich Debian GNU/Linux von Ubuntu? Nennen Sie zwei Aspekte.

2. Welche sind die gängigsten Umgebungen/Plattformen, in denen Linux eingesetzt wird? Nennen Sie drei verschiedene Umgebungen/Plattformen und nennen Sie eine Distribution, die sich für alle eignet.

3. Sie planen die Installation einer Linux-Distribution in einer neuen Umgebung. Nennen Sie vier Dinge, die Sie bei der Auswahl der Distribution berücksichtigen sollten.

4. Nennen Sie — abgesehen von Smartphones — drei Geräte, auf denen das Android-Betriebssystem läuft.

5. Erklären Sie drei wesentliche Vorteile von Cloud Computing.

Offene Übungen

1. Welche Distributionen sind in Bezug auf Kosten und Leistung am besten für ein Unternehmen geeignet, das darauf abzielt, die Lizenzkosten zu senken und gleichzeitig die Leistung auf dem höchsten Niveau zu halten? Erklären Sie, warum.

2. Was sind die Hauptvorteile des Raspberry Pi und welche Funktionen kann er in Unternehmen erfüllen?

3. Welche Distributionen bieten Amazon Cloud Services und Google Cloud an? Nennen Sie mindestens drei gemeinsame und zwei verschiedene.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Welche Linux-Distributionen gibt es?
- Was sind Linux Embedded Systeme?
- Wie werden Linux Embedded Systeme eingesetzt?
- Verschiedene Einsatzmöglichkeiten von Android
- Verschiedene Einsatzmöglichkeiten des Raspberry Pi
- Was ist Cloud Computing?
- Welche Rolle spielt Linux beim Cloud Computing?

Lösungen zu den geführten Übungen

1. Inwiefern unterscheidet sich Debian GNU/Linux von Ubuntu? Nennen Sie zwei Aspekte.

   Ubuntu basiert auf einer Momentaufnahme (einem sogenannten *Snapshot*) von Debian, daher gibt es viele Ähnlichkeiten. Allerdings gibt es auch erhebliche Unterschiede: Der erste ist die Eignung für Anfänger. Ubuntu wird besonders Anfängern wegen seiner Benutzerfreundlichkeit empfohlen, Debian hingegen eher fortgeschritteneren Benutzern. Der größte Unterschied liegt in der Komplexität der Benutzerkonfiguration, die Ubuntu während des Installationsprozesses nicht hat.

   Ein weiterer Unterschied liegt in der Stabilität. Debian gilt im Vergleich zu Ubuntu als stabiler. Dies liegt daran, dass Debian weniger Updates erhält und diese intensiv getestet werden. Auf der anderen Seite ermöglicht Ubuntu dem Benutzer, die neuesten Versionen von Software und neue Technologien zu verwenden.

2. Welche sind die gängigsten Umgebungen/Plattformen, in denen Linux eingesetzt wird? Nennen Sie drei verschiedene Umgebungen/Plattformen und nennen Sie eine Distribution, die sich für alle eignet.

   Gängigen Umgebungen/Plattformen sind Smartphone, Desktop und Server. Smartphones nutzen Distributionen wie Android. Auf Desktop und Server kommen alle Distributionen zum Einsatz, die sich am besten für Typ und Funktion der jeweiligen Maschine eignen, von Debian und Ubuntu bis CentOS und Red Hat Enterprise Linux.

3. Sie planen die Installation einer Linux-Distribution in einer neuen Umgebung. Nennen Sie vier Dinge, die Sie bei der Auswahl der Distribution berücksichtigen sollten.

   Bei der Auswahl einer Distribution sollten vor allem Kosten, Leistung, Skalierbarkeit, Stabilität und der Hardwarebedarf des Systems berücksichtigt werden.

4. Nennen Sie — abgesehen von Smartphones — drei Geräte, auf denen das Android-Betriebssystem läuft.

   Andere Geräte, auf denen Android läuft, sind Smart TVs, Tablet-Computer, Android Auto und Smartwatches.

5. Erklären Sie drei wesentliche Vorteile von Cloud Computing.

   Die Hauptvorteile von Cloud Computing sind Flexibilität, einfache Wiederherstellung und niedrige Nutzungskosten. Cloud-basierte Services sind je nach Geschäftsanforderungen einfach zu implementieren und zu skalieren. Es hat einen großen Vorteil bei Backup- und

Wiederherstellungslösungen, da es Unternehmen ermöglicht, Vorfälle schneller und mit geringeren Auswirkungen zu beheben. Darüber hinaus reduziert es die Betriebskosten, da es ermöglicht, nur für die Ressourcen zu zahlen, die ein Unternehmen in einem subskriptionsbasierten Modell verwendet.

Lösungen zu den offenen Übungen

1. Welche Distributionen sind in Bezug auf Kosten und Leistung am besten für ein Unternehmen geeignet, das darauf abzielt, die Lizenzkosten zu senken und gleichzeitig die Leistung auf dem höchsten Niveau zu halten? Erklären Sie, warum.

   Eine der am besten geeigneten Distributionen für Unternehmen ist CentOS. Es umfasst alle Red Hat-Produkte der kommerziellen Variante des Betriebssystems, steht aber frei zur Verfügung. Ebenso garantieren Ubuntu LTS-Releases den Support für einen längeren Zeitraum. Die stabilen Versionen von Debian GNU/Linux werden auch häufig in Unternehmensumgebungen verwendet.

2. Was sind die Hauptvorteile des Raspberry Pi und welche Funktionen kann er in Unternehmen erfüllen?

   Raspberry Pi ist klein, arbeitet aber wie ein normaler Computer. Darüber hinaus ist er kostengünstig und bewältigt Web-Traffic und viele andere Aufgaben. Er kann als Server, Firewall und als Mainboard für Roboter und viele andere kleine Geräte dienen.

3. Welche Distributionen bieten Amazon Cloud Services und Google Cloud an? Nennen Sie mindestens drei gemeinsame und zwei verschiedene.

   Amazon und Google Cloud Services gemeinsame Dsitributionen sind Ubuntu, CentOS und Red Hat Enterprise Linux. Jeder Cloud-Provider bietet auch spezifische Distributionen an, die der andere nicht anbietet. Amazon hat Amazon Linux und Kali Linux, während Google etwa FreeBSD und Windows Server anbietet.

1.2 Die wichtigsten Open-Source-Anwendungen

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 1.2

Gewichtung

2

Hauptwissensgebiete

- Desktop-Anwendungen
- Server-Anwendungen
- Entwicklungssprachen
- Paketverwaltungs-Tools und Repositories

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

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

1.2 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	1 Die Linux-Community und Karriere im Open-Source-Umfeld
Lernziel:	1.2 Die wichtigsten Open-Source-Anwendungen
Lektion:	1 von 1

Einführung

Eine Anwendung ist ein Computerprogramm, dessen Zweck nicht direkt an das Innenleben des Computers gebunden ist, sondern an Aufgaben, die vom Benutzer ausgeführt werden. Linux-Distributionen bieten viele Anwendungsmöglichkeiten für eine Vielzahl von Aufgaben, wie zum Beispiel Office-Anwendungen, Webbrowser, Multimedia-Player und -Editoren usw. Es gibt oft mehr als eine Anwendung oder ein Werkzeug, um eine bestimmte Aufgabe auszuführen. Es ist am Benutzer, die Anwendung auszuwählen, die seinen Bedürfnissen am besten entspricht.

Softwarepakete

Fast jede Linux-Distribution bietet einen vorinstallierten Satz von Standardanwendungen. Neben diesen vorinstallierten Anwendungen verfügt eine Distribution über ein Paket-Repository mit einer riesigen Sammlung von Anwendungen, die über ihren *Paketmanager* installiert werden. Obwohl die verschiedenen Distributionen etwa dieselben Anwendungen anbieten, gibt es für verschiedene Distributionen verschiedene Paketverwaltungssysteme. Debian, Ubuntu und Linux Mint verwenden etwa die Tools `dpkg`, `apt-get` und `apt` zur Installation von Softwarepaketen, allgemein bekannt als *DEB-Pakete*. Distributionen wie Red Hat, Fedora und CentOS verwenden

stattdessen `rpm`, `yum` und `dnf`, die wiederum *RPM-Pakete* installieren. Da sich die Anwendungspakete für verschiedene Distributionsfamilien voneinander unterscheiden, ist es sehr wichtig, Pakete vom richtigen Repository zu installieren, die an die jeweilige Distribution angepasst sind. Der Benutzer muss sich üblicherweise nicht um diese Details kümmern, da der Paketmanager der Distribution die richtigen Pakete, die Abhängigkeiten und künftige Updates auswählt. Abhängigkeiten sind von den Programmen benötigte Hilfspakete. Wenn eine Bibliothek beispielsweise Funktionen zur Verarbeitung von JPEG-Bildern bereitstellt, die verschiedene Programme benötigen, wird eine solche Bibliothek üblicherweise in ein eigenes Paket gepackt, von dem alle Anwendungen, die diese Bibliothek benötigen, abhängen.

Die Befehle `dpkg` und `rpm` wirken auf einzelne Paketdateien. In der Praxis werden fast alle Aufgaben der Paketverwaltung mit den Befehlen `apt-get` oder `apt` auf Systemen, die DEB-Pakete verwenden, oder mit `yum` oder `dnf` auf Systemen, die RPM-Pakete verwenden, durchgeführt. Diese Befehle arbeiten mit Paketkatalogen, können neue Pakete und deren Abhängigkeiten herunterladen und nach neueren Versionen der installierten Pakete suchen.

Paket-Installation

Angenommen Sie haben von einem Befehl namens `figlet` gehört, der vergrößerten Text auf dem Terminal ausgibt, und Sie wollen ihn ausprobieren. Sie erhalten nach Ausführung des Befehls `figlet` jedoch die folgende Meldung:

```
$ figlet
-bash: figlet: command not found
```

Das weist vermutlich darauf hin, dass das Paket nicht auf Ihrem System installiert ist. Wenn Ihre Distribution mit DEB-Paketen arbeitet, können Sie ihre Repositories mit `apt-cache search paketname` oder `apt search paketname` durchsuchen. Der Befehl `apt-cache` wird verwendet, um nach Paketen zu suchen und Informationen über verfügbare Pakete aufzulisten. Der folgende Befehl sucht nach allen Vorkommen der Zeichenfolge “figlet” in den Namen und Beschreibungen des Pakets:

```
$ apt-cache search figlet
figlet - Make large character ASCII banners out of ordinary text
```

Die Suche hat ein Paket namens *figlet* identifiziert, das dem fehlenden Befehl entspricht. Die Installation und Deinstallation eines Pakets erfordert spezielle Berechtigungen, die nur dem Systemadministrator gewährt werden: dem Benutzer namens `root`. Auf Desktop-Systemen können normale Benutzer Pakete installieren oder entfernen, indem sie den Befehl `sudo` den

Installations-/Deinstallationsbefehlen voranstellen. Dafür müssen sie ihr Passwort eingeben, um fortzufahren. Für DEB-Pakete führen Sie die Installtion mit dem Befehl `apt-get install paketname` oder `apt install paketname` aus:

```
$ sudo apt-get install figlet
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  figlet
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

An dieser Stelle wird das Paket heruntergeladen und auf dem System installiert. Alle Abhängigkeiten, die das Paket eventuell benötigt, werden ebenfalls heruntergeladen und installiert:

```
Need to get 184 kB of archives.
After this operation, 741 kB of additional disk space will be used.
Get:1 http://archive.raspbian.org/raspbian stretch/main armhf figlet armhf 2.2.5-2 [184 kB]
Fetched 184 kB in 0s (213 kB/s)
Selecting previously unselected package figlet.
(Reading database ... 115701 files and directories currently installed.)
Preparing to unpack .../figlet_2.2.5-2_armhf.deb ...
Unpacking figlet (2.2.5-2) ...
Setting up figlet (2.2.5-2) ...
update-alternatives: using /usr/bin/figlet-figlet to provide /usr/bin/figlet (figlet) in
auto mode
Processing triggers for man-db (2.7.6.1-2) ...
```

Nach Abschluss des Downloads werden alle Dateien an die richtigen Stellen kopiert und weitere Konfigurationen durchgeführt. Anschließend ist der Befehl verfügbar:

```
$ figlet Awesome!
    _                                          _
   / \__      _____  ___  ___  _ __ ___   ___| |
  / _ \ \ /\ / / _ \/ __|/ _ \| '_ ` _ \ / _ \ |
 / ___ \ V  V /  __/\__ \ (_) | | | | | |  __/_|
/_/   \_\_/\_/ \___||___/\___/|_| |_| |_|\___(_)
```

In Distributionen, die auf RPM-Paketen basieren, erfolgt die Suche mit `yum search`

`paketname` oder `dnf search paketname`. Nehmen wir an, Sie möchten einen etwas lockeren Text, illustriert mit einer Kuh, anzeigen, aber Sie sind sich nicht sicher, welches Paket diese Aufgabe erfüllen kann. Wie bei den DEB-Paketen akzeptieren auch die RPM-Suchbefehle beschreibende Begriffe

```
$ yum search speaking cow
Last metadata expiration check: 1:30:49 ago on Tue 23 Apr 2019 11:02:33 PM -03.
==================== Name & Summary Matched: speaking, cow ====================
cowsay.noarch : Configurable speaking/thinking cow
```

Nachdem ein geeignetes Paket im Repository gefunden wurde, kann es mit `yum install paketname` oder `dnf install paketname` installiert werden:

```
$ sudo yum install cowsay
Last metadata expiration check: 2:41:02 ago on Tue 23 Apr 2019 11:02:33 PM -03.
Dependencies resolved.
================================================================================
 Package         Arch           Version               Repository         Size
================================================================================
Installing:
 cowsay          noarch         3.04-10.fc28          fedora             46 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 46 k
Installed size: 76 k
Is this ok [y/N]: y
```

Auch hier wird das gewünschte Paket mit all seinen möglichen Abhängigkeiten heruntergeladen und installiert:

```
Downloading Packages:
cowsay-3.04-10.fc28.noarch.rpm                  490 kB/s |  46 kB     00:00
--------------------------------------------------------------------------------
Total                                            53 kB/s |  46 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
```

```
Running transaction
Preparing        :                                                        1/1
Installing       : cowsay-3.04-10.fc28.noarch                             1/1
Running scriptlet: cowsay-3.04-10.fc28.noarch                             1/1
Verifying        : cowsay-3.04-10.fc28.noarch                             1/1

Installed:
cowsay.noarch 3.04-10.fc28

Complete!
```

Der Befehl `cowsay` macht genau das, was sein Name verspricht:

```
$ cowsay "Brought to you by yum"
 _________________________
< Brought to you by yum >
 -------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Obwohl sie nutzlos erscheinen mögen, bieten die Befehle `figlet` und `cowsay` eine Möglichkeit, die Aufmerksamkeit anderer Benutzer auf relevante Informationen zu lenken.

Entfernen von Paketen

Die Befehle zum Installieren von Paketen dienen auch dazu, sie zu entfernen. Alle Befehle akzeptieren das Schlüsselwort `remove`, um ein installiertes Paket zu deinstallieren: `apt-get remove paketname` oder `apt remove paketname` für DEB-Pakete und `yum remove paketname` oder `dnf remove paketname` für RPM-Pakete. Der Befehl `sudo` wird auch benötigt, um das Entfernen durchzuführen. Um etwa das zuvor installierte Paket *figlet* aus einer DEB-basierten Distribution zu entfernen:

```
$ sudo apt-get remove figlet
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  figlet
```

```
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 741 kB disk space will be freed.
Do you want to continue? [Y/n] Y
```

Nach der Bestätigung des Vorgangs wird das Paket aus dem System gelöscht:

```
(Reading database ... 115775 files and directories currently installed.)
Removing figlet (2.2.5-2) ...
Processing triggers for man-db (2.7.6.1-2) ...
```

Ähnlich ist die Vorgehensweise auf einem RPM-basierten System. Um das zuvor installierte Paket *cowsay* aus einer RPM-basierten Distribution zu entfernen:

```
$ sudo yum remove cowsay
Dependencies resolved.
================================================================================
 Package          Arch             Version                Repository         Size
================================================================================
Removing:
 cowsay           noarch           3.04-10.fc28           @fedora            76 k

Transaction Summary
================================================================================
Remove  1 Package

Freed space: 76 k
Is this ok [y/N]: y
```

Auch hier wird eine Bestätigung angefordert und das Paket aus dem System gelöscht:

```
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Erasing          : cowsay-3.04-10.fc28.noarch                             1/1
  Running scriptlet: cowsay-3.04-10.fc28.noarch                             1/1
  Verifying        : cowsay-3.04-10.fc28.noarch                             1/1

Removed:
```

```
  cowsay.noarch 3.04-10.fc28

Complete!
```

Die Konfigurationsdateien der entfernten Pakete bleiben auf dem System erhalten, so dass sie bei einer zukünftigen Neuinstallation des Pakets wieder verwendet werden können.

Office-Anwendungen

Office-Anwendungen werden für die Bearbeitung von Dateien wie Texten, Präsentationen, Tabellenkalkulationen und anderen in einer Büroumgebung gebräuchlichen Formaten verwendet. Diese Anwendungen sind üblicherweise in Sammlungen organisiert, die als *Office Suites* bezeichnet werden.

Lange Zeit war *OpenOffice.org* die unter Linux meistgenutzte Office Suite. OpenOffice.org war eine Open-Source-Version der von *Sun Microsystems* herausgegebenen *StarOffice Suite*. Einige Jahre später wurde Sun von der *Oracle Corporation* übernommen, die das Projekt wiederum an die *Apache Foundation* übertrug, und OpenOffice.org wurde in *Apache OpenOffice* umbenannt. In der Zwischenzeit veröffentlichte die *Document Foundation* auf derselben Code-Basis eine weitere Office Suite namens *LibreOffice*.

Die beiden Projekte haben die gleichen Grundfunktionen und sind kompatibel zu den Dokumentenformaten von *Microsoft Office*. Das bevorzugte Dokumentenformat ist jedoch das *Open Document Format*, ein vollständig offenes und ISO-standardisiertes Dateiformat. Die Verwendung von ODF stellt sicher, dass Dokumente zwischen Betriebssystemen und Anwendungen verschiedener Anbieter, wie beispielsweise Microsoft Office, übertragen werden können. Die Hauptanwendungen von OpenOffice/LibreOffice sind:

Writer

Textverarbeitung

Calc

Tabellenkalkulation

Impress

Präsentationen

Draw

Vektorzeichnung

Math

Mathematische Formeln

Base

Datenbank

Sowohl LibreOffice als auch Apache OpenOffice sind Open Source Software, aber LibreOffice ist unter LGPLv3 lizenziert, während Apache OpenOffice unter der Apache License 2.0 steht. Der Lizenzunterschied hat zur Folge, dass LibreOffice Verbesserungen von Apache OpenOffice übernehmen kann, aber Apache OpenOffice keine Verbesserungen von LibreOffice. Dies und eine aktivere Entwicklergemeinschaft sind die Gründe, dass die meisten Distributionen LibreOffice als Standard-Office-Suite nutzen.

Webbrowser

Den meisten Benutzern dient ein Computer vor allem dazu, Zugang zum Internet zu gewährleisten. Heutzutage können Webseiten als vollwertige App fungieren, mit dem Vorteil, dass sie von überall her zugänglich sind, ohne zusätzliche Software installieren zu müssen. Das macht den Webbrowser zur wichtigsten Anwendung des Betriebssystems, zumindest für den durchschnittlichen Benutzer.

TIP

Eine der besten Quellen, um mehr über Webentwicklung zu erfahren, sind die MDN Web Docs, verfügbar unter https://developer.mozilla.org/. Die von Mozilla betreute Website ist voll von Tutorials für Anfänger und Referenzmaterialien zu den neuesten Webtechnologien.

Die wichtigsten Webbrowser im Linux-Umfeld sind *Google Chrome* und *Mozilla Firefox*. Chrome ist ein von Google gepflegter Webbrowser, basiert aber auf dem Open Source Browser *Chromium*, der über den Paketmanager der Distribution installiert werden kann und vollständig mit Chrome kompatibel ist. Firefox wird von Mozilla, einer gemeinnützigen Organisation, verwaltet und ist ein Browser, dessen Ursprünge mit Netscape verbunden sind, dem ersten beliebten Webbrowser, der das Open-Source-Modell verwendet. Die Mozilla Foundation ist tief in der Entwicklung offener Standards verwurzelt, die dem modernen Web zugrundeliegen.

Mozilla entwickelt auch andere Anwendungen, wie den E-Mail-Client *Thunderbird*. Viele Benutzer entscheiden sich für Webmail anstelle einer dedizierten E-Mail-Anwendung, aber ein Client wie Thunderbird bietet zusätzliche Funktionen und integriert sich am besten mit anderen Anwendungen auf dem Desktop.

Multimedia

Im Vergleich zu den verfügbaren Webanwendungen sind Desktop-Anwendungen nach wie vor die beste Option für die Erstellung von Multimedia-Inhalten. Multimedia-bezogene Aktivitäten wie Video-Rendering erfordern oft viele Systemressourcen, die am besten von einer lokalen Desktop-Anwendung verwaltet werden. Einige der beliebtesten Mulitmedia-Anwendungen unter Linux und ihre Einsatzbereiche sind:

Blender

Ein 3D-Renderer zur Erstellung von Animationen, mit dem auch 3D-Objekte exportiert werden können, die auf einem 3D-Drucker gedruckt werden sollen.

GIMP

Ein vollwertiges Bildbearbeitungsprogramm, das mit *Adobe Photoshop* vergleichbar ist, aber über eigene Konzepte und Werkzeuge verfügt. GIMP kann die meisten Bitmap-Dateien wie JPEG, PNG, GIF, TIFF und viele andere erstellen, bearbeiten und speichern.

Inkscape

Ein Vektorgrafik-Editor, ähnlich wie *Corel Draw* oder *Adobe Illustrator*. Das Standardformat von Inkscape ist SVG, ein offener Standard für Vektorgrafiken. SVG-Dateien können von jedem Webbrowser geöffnet werden und sind als Vektorgrafiken in flexiblen Webseiten-Layouts einsetzbar.

Audacity

Ein Audio-Editor, mit dem man filtern, Effekte anwenden und zwischen vielen verschiedenen Audioformaten wie MP3, WAV, OGG, FLAC etc. konvertieren kann.

ImageMagick

ImageMagick ist ein Kommandozeilenwerkzeug zur Konvertierung und Bearbeitung der meisten Bilddateitypen. Es dient auch dazu, PDF-Dokumente aus Bilddateien zu erstellen und umgekehrt.

Es gibt auch zahlreiche Anwendungen zur Multimedia-Wiedergabe. Die beliebteste Anwendung für die Video-Wiedergabe ist *VLC*, aber einige Benutzer bevorzugen andere Alternativen, wie *smplayer*. Die lokale Musikwiedergabe hat auch viele Optionen, wie *Audacious*, *Banshee* und *Amarok*, die auch eine Sammlung lokaler Sounddateien verwalten können.

Server-Programme

Wenn ein Webbrowser eine Seite von einer Website lädt, verbindet er sich tatsächlich mit einem

entfernten Computer und fragt nach einer bestimmten Information. In diesem Szenario wird der Computer, auf dem der Webbrowser läuft, als *Client* bezeichnet, der entfernte Computer als *Server*.

Der Server-Rechner, der ein gewöhnlicher Desktop-Computer oder spezielle Hardware sein kann, benötigt ein spezifisches Programm, das die bereitzustellenden Informationen verwaltet. Was die Bereitstellung von Webseiten betrifft, so setzen die meisten Server auf der ganzen Welt Open-Source-Serverprogramme ein. Dieses spezielle Serverprogramm wird als *HTTP-Server* bezeichnet (HTTP steht für *Hyper Text Transfer Protocol*), und die beliebtesten sind *Apache*, *Nginx* und *lighttpd*.

Auch einfache Webseiten können viele Anfragen erfordern, zum Beispiel einfache Dateien — sog. statische Inhalte — oder dynamische Inhalte, die aus verschiedenen Quellen zusammengefügt werden. Die Aufgabe eines HTTP-Servers besteht darin, alle angeforderten Daten zu sammeln und an den Browser zurückzusenden, der dann den Inhalt, wie er durch das empfangene HTML-Dokument (HTML steht für *Hyper Text Markup Language*) und andere unterstützende Dateien definiert ist, anordnet. Daher umfasst das Rendern einer Webseite Operationen sowohl auf der Serverseite wie auch auf der Clientseite. Beide Seiten können benutzerdefinierte Skripte verwenden, um bestimmte Aufgaben zu erfüllen. Auf der HTTP-Serverseite ist es durchaus üblich, die Skriptsprache PHP zu verwenden. JavaScript ist die Skriptsprache, die auf der Clientseite (dem Webbrowser) verwendet wird.

Serverprogramme können alle Arten von Informationen bereitstellen: Es ist nicht ungewöhnlich, dass ein Serverprogramm Informationen von anderen Serverprogrammen anfordert, z.B. wenn ein HTTP-Server Informationen von einem Datenbankserver benötigt.

Wenn beispielsweise eine dynamische Seite angefordert wird, fragt der HTTP-Server in der Regel eine Datenbank ab, um alle erforderlichen Informationen zu sammeln, und sendet den dynamischen Inhalt an den Client zurück. Ähnlich verhält es sich, wenn sich ein Benutzer auf einer Website registriert: Der HTTP-Server sammelt die vom Client gesendeten Daten und speichert sie in einer Datenbank.

Eine Datenbank ist ein organisierter Satz von Informationen. Ein Datenbankserver speichert Inhalte in formatierter Form und ermöglicht das Lesen, Schreiben und Verknüpfen großer Datenmengen in hoher Geschwindigkeit und Zuverlässigkeit. Open-Source-Datenbankserver werden in vielen Anwendungen eingesetzt, nicht nur im Internet, sondern auch in lokalen Anwendungen, die Daten durch Verbindung zu einem lokalen Datenbankserver speichern können. Der häufigste Typ von Datenbank ist die *relationale Datenbank*, in der die Daten in vordefinierten Tabellen organisiert sind. Die beliebtesten relationalen Open-Source-Datenbanken sind *MariaDB* (entstanden aus *MySQL*) und *PostgreSQL*.

Data Sharing

In lokalen Netzwerken, wie sie in Büros und zu Hause zu finden sind, ist es wünschenswert, dass Computer nicht nur auf das Internet zugreifen, sondern auch miteinander kommunizieren. Manchmal fungiert ein Computer als Server, manchmal derselbe Computer als Client. Das ist zum Beispiel notwendig, wenn man auf Dateien auf einem anderen Computer im Netzwerk zugreifen möchte — zum Beispiel auf eine auf einem Desktop-Computer gespeicherte Datei von einem tragbaren Gerät aus — ohne die Mühe, sie auf ein USB-Laufwerk oder dergleichen zu kopieren.

Zwischen Linux-Maschinen wird häufig NFS (*Network File System*) verwendet. Das NFS-Protokoll ist der Standardweg, um Dateisysteme in Netzwerken zu teilen, die nur mit Unix/Linux-Maschinen ausgestattet sind. NFS ermöglicht es einem Computer, eines oder mehrere seiner Verzeichnisse mit bestimmten Computern im Netzwerk zu teilen, damit er Dateien in diesen Verzeichnissen lesen und schreiben kann. NFS kann sogar verwendet werden, um den Verzeichnisbaum eines ganzen Betriebssystems mit Clients zu teilen, die es zum Booten verwenden. Diese Computer, die sogenannten *Thin Clients*, werden meist in großen Netzwerken eingesetzt, um die Wartung jedes einzelnen Betriebssystems auf jeder Maschine zu vermeiden.

Wenn andere Arten von Betriebssystemen an das Netzwerk angeschlossen sind, empfiehlt es sich, ein für alle verständliches Data-Sharing-Protokoll zu verwenden. Diese Aufgaben erfüllt *Samba*. Samba implementiert ein Protokoll für den Dateiaustausch über das Netzwerk, das ursprünglich für das Windows-Betriebssystem entwickelt wurde, heute aber mit allen gängigen Betriebssystemen kompatibel ist. Mit Samba können Computer im lokalen Netzwerk nicht nur Dateien, sondern auch Drucker teilen.

In einigen lokalen Netzwerken wird die Berechtigung, die bei der Anmeldung auf einer Workstation erteilt wird, von einem zentralen Server, dem *Domänencontroller* , vergeben. Er verwaltet den Zugriff auf verschiedene lokale und entfernte Ressourcen. Der Domänencontroller ist ein Dienst aus dem *Active Directory* von Microsoft. Linux-Workstations können sich mit einem Domänencontroller über Samba oder ein Authentifizierungssubsystem namens *SSSD* verbinden. Ab Version 4 kann Samba auch als Domänencontroller in heterogenen Netzwerken fungieren.

Wenn es darum geht, eine Cloud-Computing-Lösung zu implementieren, die verschiedene Methoden des webbasierten Datenaustauschs bietet, sollten zwei Alternativen in Betracht gezogen werden: *ownCloud* und *Nextcloud*. Beide Projekte sind sehr ähnlich, da Nextcloud ein Spin-off von ownCloud ist, was bei Open-Source-Projekten nicht ungewöhnlich ist. Solche Spin-offs bezeichnet man als *Fork*. Beide bieten die gleichen Grundfunktionen: File Sharing und Synchronisation, kollaborative Arbeitsbereiche, Kalender, Kontakte und E-Mails — alles über Desktop-, Mobil- und Webschnittstellen. Nextcloud bietet auch private Audio-/Videokonferenzen an, während sich ownCloud mehr auf die gemeinsame Nutzung von Dateien und die Integration mit Software von Drittanbietern konzentriert. Viele weitere Funktionen sind als Plugins verfügbar, die bei Bedarf

später aktiviert werden können.

Sowohl ownCloud als auch Nextcloud bieten eine kostenpflichtige Version mit zusätzlichen Funktionen und erweitertem Support an, die sich von anderen kommerziellen Lösungen dadurch unterscheidet, dass sie Nextcloud oder ownCloud kostenlos auf einem privaten Server installieren können, ohne sensible Daten auf einem unbekannten Server vorzuhalten. Da alle Dienste auf HTTP-Kommunikation basieren und in PHP geschrieben sind, muss die Installation auf einem zuvor konfigurierten Webserver wie Apache erfolgen. Wenn Sie die Installation von ownCloud oder Nextcloud auf Ihrem eigenen Server in Betracht ziehen, stellen Sie sicher, dass HTTPS auch alle Verbindungen zu Ihrer Cloud verschlüsselt.

Netzwerkadministration

Kommunikation zwischen Computern ist nur möglich, wenn das Netzwerk ordnungsgemäß funktioniert. Die Netzwerkkonfiguration erfolgt in der Regel durch eine Reihe von Programmen, die auf dem Router laufen und für die Einrichtung und Überprüfung der Netzwerkfähigkeit zuständig sind. Voraussetzung dafür sind zwei grundlegende Netzwerkdienste: *DHCP* (*Dynamic Host Configuration Protocol*) und *DNS* (*Domain Name System*).

DHCP ist dafür verantwortlich, dem Host eine IP-Adresse zuzuweisen, wenn ein Netzwerkkabel angeschlossen ist oder wenn das Gerät in ein drahtloses Netzwerk eintritt. Bei der Verbindung mit dem Internet stellt der DHCP-Server des ISP dem anfragenden Gerät eine IP-Adresse zur Verfügung. Ein DHCP-Server ist auch in lokalen Netzwerken sehr nützlich, um allen angeschlossenen Geräten automatisch IP-Adressen zuzuweisen. Wenn DHCP nicht konfiguriert ist oder nicht ordnungsgemäß funktioniert, muss die IP-Adresse jedes mit dem Netzwerk verbundenen Geräts manuell konfiguriert werden. Es ist nicht sinnvoll, die IP-Adressen in großen Netzwerken oder sogar in kleinen Netzwerken manuell einzustellen; deshalb verfügen die meisten Netzwerkrouter über einen vorkonfigurierten DHCP-Server.

Die IP-Adresse wird benötigt, um mit einem anderen Gerät in einem IP-Netzwerk zu kommunizieren, aber Domainnamen wie `www.lpi.org` sind viel einfacher zu merken als eine IP-Adresse wie `203.0.113.165`. Der Domainname allein reicht jedoch nicht aus, um die Kommunikation über das Netzwerk herzustellen, weshalb der Domainname von einem DNS-Server in eine IP-Adresse übersetzt werden muss: Die IP-Adresse des DNS-Servers wird vom DHCP-Server des ISP bereitgestellt und von allen angeschlossenen Systemen zur Übersetzung von Domainnamen in IP-Adressen verwendet.

Sowohl die DHCP- als auch die DNS-Einstellungen lassen sich über die vom Router bereitgestellte Webschnittstelle ändern. So ist es zum Beispiel möglich, die IP-Zuweisung nur auf bekannte Geräte zu beschränken oder bestimmten Maschinen eine feste IP-Adresse zuzuordnen. Auch der vom ISP bereitgestellte Standard-DNS-Server lässt sich darüber ändern. Einige DNS-Server von

Drittanbietern, wie sie von Google oder OpenDNS bereitgestellt werden, liefern bisweilen schnellere Antworten und bieten zusätzliche Funktionen.

Programmiersprachen

Alle Computerprogramme (Client- und Serverprogramme, Desktop-Anwendungen und das Betriebssystem selbst) werden in einer oder mehreren Programmiersprachen erstellt, wobei es sich bei den Programmen um eine einzelne Datei oder ein komplexes System aus Hunderten von Dateien handeln kann, die das Betriebssystem als Befehlssequenz behandelt, die vom Prozessor und anderen Geräten interpretiert und ausgeführt wird.

Es gibt zahlreiche Programmiersprachen für sehr unterschiedliche Zwecke, und Linux-Systeme stellen viele davon bereit. Da Open-Source-Software auch die Quellen der Programme umfasst, bieten Linux-Systeme Entwicklern perfekte Voraussetzungen, um Software zu verstehen und nach ihren eigenen Bedürfnissen zu modifizieren oder zu erstellen.

Jedes Programm beginnt als Textdatei, dem *Quellcode*. Dieser Quellcode ist in einer mehr oder weniger menschenlesbaren Sprache geschrieben, die beschreibt, was das Programm tut. Ein Computerprozessor kann diesen Code nicht direkt ausführen; bei *kompilierten Sprachen* wird der Quellcode daher in eine *binäre Datei* umgewandelt, die dann vom Computer ausgeführt werden kann. Ein Programm namens *Compiler* ist für die Konvertierung von Quellcode in eine ausführbare Form verantwortlich. Da die kompilierte Binärdatei für einen Prozessortyp spezifisch ist, muss das Programm möglicherweise neu kompiliert werden, um auf einem anderen Computertyp ausgeführt zu werden.

Bei *interpretierten Sprachen* muss das Programm nicht vorher kompiliert werden. Stattdessen liest ein *Interpreter* den Quellcode und führt bei jedem Programmstart seine Befehle aus. Das macht die Entwicklung einfacher und schneller, aber dafür sind interpretierte Programme tendenziell langsamer als kompilierte.

Hier einige der beliebtesten Programmiersprachen:

JavaScript

JavaScript ist eine Programmiersprache, die vor allem auf Webseiten verwendet wird. Ursprünglich waren JavaScript-Anwendungen sehr einfach, wie Formularvalidierungsroutinen. Heute wird JavaScript zur Erstellung sehr komplexer Anwendungen nicht nur im Web, sondern auch auf Servern und mobilen Geräten eingesetzt.

C

Die Programmiersprache C ist eng mit Betriebssystemen, insbesondere Unix, verbunden, wird aber verwendet, um jede Art von Programm auf fast jede Art von Gerät zu schreiben. Die

großen Vorteile von C sind Flexibilität und Geschwindigkeit. Der gleiche in C geschriebene Quellcode kann auf verschiedenen Plattformen und Betriebssystemen mit wenig oder gar keiner Änderung kompiliert werden. Nach der Kompilierung läuft das Programm jedoch nur auf dem Zielsystem.

Java

Der Hauptaspekt von Java ist, dass Programme, die in dieser Sprache geschrieben wurden, portabel sind, was bedeutet, dass das gleiche Programm auf verschiedenen Betriebssystemen ausgeführt werden kann. Trotz des Namens hat Java nichts mit JavaScript zu tun.

Perl

Perl ist eine Programmiersprache, die meist zur Verarbeitung von Textinhalten verwendet wird. Ihre besondere Stärke sind reguläre Ausdrücke, was Perl zu einer Sprache insbesondere für Textfilterung und -analyse macht.

Shell

Die Shell, insbesondere die Bash-Shell, ist nicht nur eine Programmiersprache, sondern eine interaktive Schnittstelle zur Ausführung anderer Programme. Shell-Programme, *Shell-Skripte* genannt, können komplexe oder sich wiederholende Aufgaben in der Kommandozeilenumgebung automatisieren.

Python

Python ist eine sehr beliebte Programmiersprache bei Studenten und Fachleuten, die nichts direkt mit Informatik zu tun haben. Obwohl Python über fortgeschrittene Funktionen verfügt, ist es aufgrund seines einfachen Ansatzes ein guter Einstieg in die Programmierung.

PHP

PHP wird meist als serverseitige Skriptsprache zur Generierung von Inhalten für das Web verwendet. Die meisten Online-HTML-Seiten sind keine statischen Dateien, sondern dynamische Inhalte, die vom Server aus verschiedenen Quellen, wie z.B. Datenbanken, generiert werden. PHP-Programme — manchmal auch nur PHP-Seiten oder PHP-Skripte genannt — dienen oft dazu, diese Inhalte zu generieren. Der Begriff LAMP kommt aus der Kombination eines Linux-Betriebssystems, eines Apache HTTP-Servers, einer MySQL (oder MariaDB) Datenbank und PHP-Programmierung. LAMP-Server sind eine sehr beliebte Lösung für den Betrieb von Webservern. Neben PHP können auch alle zuvor beschriebenen Programmiersprachen zur Implementierung solcher Anwendungen verwendet werden.

C und Java sind kompilierte Sprachen, wobei der in C geschriebene Quellcode in binären Maschinencode umgewandelt wird, während der Java-Quellcode in *Bytecode* umgewandelt wird, der in einer speziellen Softwareumgebung namens *Java Virtual Machine* ausgeführt wird.

JavaScript, Perl, Shell-Skript, Python und PHP sind interpretierte Sprachen, die auch als *Skriptsprachen* bezeichnet werden.

Geführte Übungen

1. Identifizieren Sie für jeden der folgenden Befehle, ob er zum *Debian-Paketverwaltungssystem* oder zum *Red Hat-Paketverwaltungssystem* gehört:

`dpkg`	
`rpm`	
`apt-get`	
`yum`	
`dnf`	

2. Welcher Befehl könnte verwendet werden, um Blender auf Ubuntu zu installieren? Wie kann das Programm nach der Installation ausgeführt werden?

3. Welche Anwendung aus der LibreOffice Suite dient der elektronischen Tabellenkalkulation?

4. Welcher Open-Source-Webbrowser bildet die Grundlage für die Entwicklung von Google Chrome?

5. SVG ist ein offener Standard für Vektorgrafiken. Welche ist die beliebteste Anwendung zum Bearbeiten von SVG-Dateien in Linux-Systemen?

6. Schreiben Sie für jedes der folgenden Dateiformate den Namen einer Anwendung, die die entsprechende Datei öffnen und bearbeiten kann:

`png`	
`doc`	
`xls`	
`ppt`	
`wav`	

7. Welches Softwarepaket ermöglicht den Dateiaustausch zwischen Linux- und Windows-Rechnern über das lokale Netzwerk?

Offene Übungen

1. Sie wissen, dass Konfigurationsdateien auch dann erhalten bleiben, wenn das zugehörige Paket aus dem System entfernt wird. Wie können Sie das Paket namens *cups* und seine Konfigurationsdateien automatisch aus einem DEB-basierten System entfernen?

2. Angenommen Sie haben viele TIFF-Bilddateien und möchten diese in JPEG konvertieren. Welches Softwarepaket könnte verwendet werden, um diese Dateien direkt auf der Kommandozeile zu konvertieren?

3. Welches Softwarepaket müssen Sie installieren, um Microsoft-Word-Dokumente öffnen zu können, die Ihnen von einem Windows-Benutzer zugesandt wurden?

4. Jedes Jahr veranstaltet linuxquestions.org eine Umfrage über die beliebtesten Linux-Anwendungen. Finden Sie unter https://www.linuxquestions.org/questions/2018-linuxquestions-org-members-choice-awards-128/ heraus, welche Desktop-Anwendungen bei erfahrenen Linux-Anwendern am beliebtesten sind.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Die Paketmanagementsysteme, die in den wichtigsten Linux-Distributionen verwendet werden
- Open-Source-Anwendungen, die gängige Dateiformate bearbeiten können
- Die Serverprogramme, die vielen wichtigen Internet- und lokalen Netzwerkdiensten zugrundeliegen
- Verbreitete Programmiersprachen und deren Anwendungen

Lösungen zu den geführten Übungen

1. Identifizieren Sie für jeden der folgenden Befehle, ob er zum *Debian-Paketverwaltungssystem* oder zum *Red Hat-Paketverwaltungssystem* gehört:

`dpkg`	Debian-Paketverwaltungssystem
`rpm`	Red Hat-Paketverwaltungssystem
`apt-get`	Debian-Paketverwaltungssystem
`yum`	Red Hat-Paketverwaltungssystem
`dnf`	Red Hat-Paketverwaltungssystem

2. Welcher Befehl könnte verwendet werden, um Blender auf Ubuntu zu installieren? Wie kann das Programm nach der Installation ausgeführt werden?

   Der Befehl `apt-get install blender`. Der Paketname sollte in Kleinbuchstaben angegeben werden. Das Programm kann direkt vom Terminal aus mit dem Befehl `blender` oder durch Auswahl aus dem Anwendungsmenü ausgeführt werden.

3. Welche Anwendung aus der LibreOffice Suite dient der elektronischen Tabellenkalkulation?

   Calc

4. Welcher Open-Source-Webbrowser bildet die Grundlage für die Entwicklung von Google Chrome?

   Chromium

5. SVG ist ein offener Standard für Vektorgrafiken. Welche ist die beliebteste Anwendung zum Bearbeiten von SVG-Dateien in Linux-Systemen?

   Inkscape

6. Schreiben Sie für jedes der folgenden Dateiformate den Namen einer Anwendung, die die entsprechende Datei öffnen und bearbeiten kann:

`png`	Gimp
`doc`	LibreOffice Writer
`xls`	LibreOffice Calc
`ppt`	LibreOffice Impress

`wav`	Audacity

7. Welches Softwarepaket ermöglicht den Dateiaustausch zwischen Linux- und Windows-Rechnern über das lokale Netzwerk?

   Samba

Lösungen zu den offenen Übungen

1. Sie wissen, dass Konfigurationsdateien auch dann erhalten bleiben, wenn das zugehörige Paket aus dem System entfernt wird. Wie können Sie das Paket namens *cups* und seine Konfigurationsdateien automatisch aus einem DEB-basierten System entfernen?

   `apt-get purge cups`

2. Angenommen Sie haben viele TIFF-Bilddateien und möchten diese in JPEG konvertieren. Welches Softwarepaket könnte verwendet werden, um diese Dateien direkt auf der Kommandozeile zu konvertieren?

   ImageMagick

3. Welches Softwarepaket müssen Sie installieren, um Microsoft-Word-Dokumente öffnen zu können, die Ihnen von einem Windows-Benutzer zugesandt wurden?

   LibreOffice oder OpenOffice

4. Jedes Jahr veranstaltet linuxquestions.org eine Umfrage über die beliebtesten Linux-Anwendungen. Finden Sie unter https://www.linuxquestions.org/questions/2018-linuxquestions-org-members-choice-awards-128/ heraus, welche Desktop-Anwendungen bei erfahrenen Linux-Anwendern am beliebtesten sind.

   Browser: Firefox. E-Mail-Client: Thunderbird. Mediaplayer: VLC. Rastergrafik-Editor: GIMP.

1.3 Open-Source-Software und -Lizenzen

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 1.3

Gewichtung

1

Hauptwissensgebiete

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

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- Copyleft, Permissive
- GPL, BSD, Creative Commons
- Freie Software, Open Source Software, FOSS, FLOSS
- Open-Source-Geschäftsmodelle

1.3 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	1 Die Linux-Community und Karriere im Open-Source-Umfeld
Lernziel:	1.3 Open-Source-Software und -Lizenzen
Lektion:	1 von 1

Einführung

Während die Begriffe *freie Software* und *Open Source Software* weit verbreitet sind, gibt es immer noch einige Missverständnisse über ihre Bedeutung. Insbesondere das Konzept der “Freiheit” bedarf einer näheren Betrachtung. Beginnen wir mit der Definition der beiden Begriffe.

Definition von freier und Open Source Software

Kriterien für freie Software

Zunächst einmal hat “frei” im Kontext freier Software nichts mit “kostenlos” zu tun, oder wie der Gründer der *Free Software Foundation* (FSF), Richard Stallman, es prägnant ausdrückt:

> Um das Konzept zu verstehen, sollten Sie bei “frei” an “Redefreiheit” denken, nicht an “Freibier”.
>
> — Richard Stallman, Was ist freie Software?

Unabhängig davon, ob Sie für die Software bezahlen müssen oder nicht, gibt es vier Kriterien, die freie Software ausmachen. Richard Stallman beschreibt diese Kriterien als “die vier wesentlichen Freiheiten”, deren Zählung bei Null beginnt:

- “Die Freiheit, das Programm auszuführen wie man möchte, für jeden Zweck (Freiheit 0).”

  Wo, wie und zu welchem Zweck die Software eingesetzt wird, kann weder vorgeschrieben noch eingeschränkt werden.

- “Die Freiheit, die Funktionsweise des Programms zu untersuchen und eigenen Datenverarbeitungbedürfnissen anzupassen (Freiheit 1). Der Zugang zum Quellcode ist dafür Voraussetzung.”

  Jeder kann die Software nach seinen Vorstellungen und Bedürfnissen ändern. Dies wiederum setzt voraus, dass der sogenannte *Quellcode*, d.h. alle Dateien, aus denen eine Software besteht, müssen in einer für Programmierer lesbaren Form vorliegen. Und natürlich gilt dieses Recht für einen einzelnen Benutzer, der eine einzelne Funktion hinzufügen möchte, sowie für Softwareunternehmen, die komplexe Systeme wie Smartphone-Betriebssysteme oder Router-Firmware erstellen.

- “Die Freiheit, das Programm zu redistribuieren und damit Mitmenschen zu helfen (Freiheit 2). Der Zugang zum Quellcode ist dafür Voraussetzung.”

  Diese Freiheit ermutigt jeden Benutzer ausdrücklich, die Software mit anderen zu teilen. Es geht also um die größtmögliche Verbreitung und damit die größtmögliche Gemeinschaft von Nutzern und Entwicklern, die auf der Grundlage dieser Freiheiten die Software zum Wohle aller weiterentwickeln und verbessern.

- “Die Freiheit, das Programm zu verbessern und diese Verbesserungen der Öffentlichkeit freizugeben, damit die gesamte Gesellschaft davon profitiert (Freiheit 3). Der Zugang zum Quellcode ist dafür Voraussetzung.”

  Es geht also nicht nur um die Verbreitung von freier Software, sondern auch um die Verbreitung von *veränderter* freier Software. Jeder, der Änderungen an freier Software vornimmt, hat das Recht, die Änderungen anderen zugänglich zu machen. Wenn man dies tut, ist man auch dazu verpflichtet, d.h. man darf die ursprünglichen Freiheiten bei der Verbreitung der Software nicht einschränken, auch wenn man sie verändert oder erweitert hat. Wenn eine Gruppe von Entwicklern beispielsweise unterschiedliche Vorstellungen von der künftigen Ausrichtung einer Software als die ursprünglichen Autoren hat, kann sie ihren eigenen Entwicklungszweig (*Fork* genannt) abspalten und als neues Projekt fortsetzen. Aber natürlich bleiben alle Verpflichtungen im Zusammenhang mit diesen Freiheiten bestehen.

Die Betonung des Freiheitsgedankens ist auch insofern konsequent als sich jede Freiheitsbewegung *gegen* etwas richtet, nämlich einen Gegner, der die postulierten Freiheiten unterdrückt, der Software als Eigentum betrachtet und unter Verschluss halten will. Im Gegensatz zu freier Software wird solche Software als *proprietär* (*proprietary*) bezeichnet.

Open Source Software vs. freie Software

Für viele sind *freie Software* und *Open Source Software* Synonyme. Die häufig verwendete Abkürzung *FOSS* für *Free and Open Source Software* betont genau diese Gemeinsamkeit. FLOSS für *Free/Libre and Open Source Software* ist eine weitere beliebte Begriffsvariante, die den Freiheitsgedanken auch für andere Sprachräume als den englischen unmissverständlich herausstellt. Betrachtet man allerdings Entstehung und Entwicklung beider Begriffe, lohnt sich eine Differenzierung.

Der Terminus *freie Software* mit der Definition der beschriebenen vier Freiheiten geht auf Richard Stallman und das bereits 1985 — also beinahe 10 Jahre vor der Entstehung von Linux — von ihm gegründete Projekt *GNU* zurück. Der Name “GNU is not Unix” beschreibt augenzwinkernd die Intention: GNU startete als Initiative, eine technisch überzeugende, aber bis dahin allein der Kontrolle von Konzernen unterliegende Lösung — nämlich das Betriebssystem Unix — von Grund auf neu zu entwickeln, der Allgemeinheit zur Verfügung zu stellen und mit der Allgemeinheit laufend zu verbessern. Die Offenheit des Source Code war dafür lediglich eine technisch-organisatorische Notwendigkeit — in ihrem Selbstverständnis ist die Free-Software-Bewegung aber bis heute eine *soziale* und *politische* — manche sagen auch ideologische — Bewegung.

Mit dem Erfolg von Linux, den kollaborativen Möglichkeiten des Internet und den Tausenden Projekten und Firmen, die in diesem neuen Software-Kosmos entstanden, trat der soziale Aspekt immer öfter in den Hintergrund. Die Offenheit des Quellcodes wurde von einer technischen Voraussetzung selbst zu einem Definitionsmerkmal: Sobald der Quellcode einsehbar war, galt die Software als Open Source. Die sozialen Motive wichen einem eher pragmatischen Ansatz der Softwareentwicklung.

Dieses Spannungsverhältnis besteht bis heute: Freie Software und Open Source Software arbeiten an derselben Sache, mit denselben Methoden und in einer weltweiten Community aus Einzelpersonen, Projekten und Firmen. Aber da sie soz. aus verschiedenen Richtungen — nämlich einer sozialen und einer pragmatisch-technischen — zusammengefunden haben, kommt es bisweilen zu Konflikten. Diese Konflikte entstehen, wenn die Ergebnisse der gemeinsamen Arbeit nicht den ursprünglichen Zielen *beider* Bewegungen entsprechen. Das geschieht vor allem dann, wenn eine Software zwar ihre Quellen offenlegt, aber nicht zugleich die vier Freiheiten der freien Software respektiert, wenn es also bespielsweise bei der Offenlegung, bei der Veränderung oder bei der Verbindungen mit anderen Software-Komponenten Beschränkungen gibt.

Welchen Bedingungen eine Software hinischtlich Nutzung, Weitergabe und Veränderung unterliegt, bestimmt die *Lizenz*, unter der die Software steht. Und weil Anforderungen und Motive sehr unterschiedlich sein können, sind im FOSS-Bereich zahllose verschiedene Lizenzen entstanden. Aufgrund des deutlich fundamentaleren Ansatzes der Free-Software-Bewegung nimmt es nicht Wunder, dass sie viele Open-Source-Lizenzen nicht als “frei” anerkennt und darum abgelehnt. Umgekehrt ist das aufgrund des deutlich pragmatischeren Open-Source-Ansatzes naturgemäß kaum der Fall.

Werfen wir im Folgenden einen kurzen Blick auf das tatsächlich sehr komplexe Gebiet der Lizenzen.

Lizenzen

Anders als ein Kühlschrank oder ein Auto ist Software kein physisches, sondern ein digitales Produkt. Eine Firma durch Verkauf ein solches Produkt also nicht physisch übergeben — sie übergibt vielmehr die Nutzungsrechte an diesem Produkt, und der Nutzer stimmt vertraglich diesen Nutzungsrechten zu. Welche Nutzungsrechte das sind und vor allem *nicht* sind, ist in der Software-Lizenz festgehalten, und damit wird verständlich, welche Bedeutung den darin festgehaltenen Regelungen zukommt.

Während große Anbieter proprietärer Software wie Microsoft oder SAP eigene, genau auf ihre Produkte abgestimmte Lizenzen haben, waren die Verfechter freier und quelloffener Software von Anfang an um Klarheit und Allgemeingültigkeit ihrer Lizenzen bemüht, denn schließlich soll jeder Benutzer diese verstehen und gegebenenfalls selbst für seine eigenen Entwicklungen nutzen.

Es sei allerdings nicht verschwiegen, dass dieses Ideal der Einfachheit kaum zu erreichen ist, denn zu viele spezifische Anforderungen einerseits und international nicht immer kompatible Rechtsverständnisse stehen dem entgegen. Um nur ein Beispiel zu nennen: Deutsches und amerikanisches Recht unterscheiden sich fundamental im Urheberrecht. Nach deutscher Rechtsauffassung gibt es eine *Person* als *Urheber*, deren Werk ihr *geistiges Eigentum* ist. Der Urheber kann zwar die Erlaubnis zur Nutzung seines Werkes erteilen, kann aber seine Urheberschaft nicht abtreten oder aufgeben. Letzteres ist dem amerikanischen Recht. Auch hier gibt es zwar einen *Autor* (der allerdings auch eine Firma oder eine Institution sein kann), aber er verfügt lediglich über Verwertungsrechte, die er in Teilen oder vollständig übertragen und sich damit vollständig von seinem Werk lösen kann. Eine international gültige Lizenz muss vor solch unterschiedlichen Rechtsauffassungen interpretierbar sein.

Die Folge sind zahlreiche und zum Teil sehr verschiedene FOSS-Lizenzen. Und was noch schlimmer ist: Verschiedene Versionen einer Lizenz oder die Mischung verschiedener Lizenzen (innerhalb eines Projekts oder auch bei der Verbindung mehrere Projekte) sorgen bisweilen für

Verwirrung oder gar juristische Streitfälle.

Sowohl die Vertreter freier Software wie auch die Befürworter der deutlich ökonomisch orientierterten Open-Source-Bewegung schufen eigene Organisationen, die heute maßgeblich für die Formulierung von Software-Lizenzen gemäß ihren Grundsätzen zuständig sind und ihre Mitglieder bei deren Durchsetzung unterstützen.

Copyleft

Die bereits erwähnte *Free Software Foundation* (FSF) hat mit der *GNU General Public License* (GPL) eine der wichtigsten Lizenzen für freie Software formuliert, die viele Projekte, z.B. auch der Linux-Kernel, nutzen. Darüber hinaus hat sie weitere Lizenzen mit fallspezifischen Anpassungen veröffentlicht, etwa die *GNU Lesser General Public License* (LGPL), die die Kombination von freier Software mit verändertem Code regelt, wobei der Quellcode für die Veränderungen nicht veröffentlicht werden muss, die *GNU Affero General Public License* (AGPL), die den Verkauf des Zugangs zu gehosteter Software regelt, oder die *GNU Free Documentation License* (FDL), mit der sie die freiheitlichen Grundsätze auf die Dokumentation von Software ausdehnt. Darüber hinaus spricht die FSF Empfehlungen für oder gegen Lizenzen Dritter aus, und angegliederte Projekte wie GPL-Violations.org gehen beispielsweise Verdachtsfällen nach, in denen es um die Verletzung freier Lizenzen geht.

Das Prinzip, nach dem sich eine freie Lizenz auch auf veränderte Varianten der Software überträgt, bezeichnet die FSF als *Copyleft* — im Gegensatz zu dem von ihr abgelehnten Prinzip des restriktiven Copyright. Die Idee ist also, die freiheitlichen Grundsätze einer Software-Lizenz möglichst uneingeschränkt auf künftige Bearbeitungen der Software zu übertragen, um nachträgliche Restriktionen zu verhindern.

Was naheliegend und einfach klingt, führt in der Praxis jedoch zu zum Teil erheblichen Komplikationen, weshalb das Copyleft-Prinzip von Kritikern häufig auch als “viral” bezeichnet wird, da es sich auf Folgeversionen überträgt.

Aus dem Gesagten folgt zum Beispiel, dass sich zwei Software-Komponenten, die unter verschiedenen Copyleft-Lizenzen stehen, nicht miteinander kombinieren lassen, da sich ja nicht beide Lizenzen gleichzeitig auf das Folgeprodukt übertragen lassen. Das kann sogar für verschiedene Versionen derselben Lizenz gelten!

Aus diesem Grunde fassen neuere Lizenzen oder Lizenz-Versionen das Copyleft oftmals nicht mehr so rigoros. Schon die genannte *GNU Lesser General Public License* (LGPL) ist in diesem Sinne ein Zugeständnis, um freie Software überhaupt mit “unfreien” Komponenten verbinden zu können, wie es zum Beispiel häufig bei *(Programm-)Bibliotheken* (engl. *libraries*) geschieht. Bibliotheken enthalten Unterprogramme oder Routinen, die wiederum von anderen Programmen

genutzt werden. Das führt oftmals zu der Situation, dass proprietäre Software solche Routinen einer freien Bibliothek aufruft.

Eine weitere Möglichkeit zur Vermeidung von Lizenzkonflikten ist das *Dual Licensing*, bei dem eine Software unter verschiedenen Lizenzen steht, z.B. einer freien und einer proprietären. Ein typischer Anwendungsfall ist eine kostenlose Version einer Software, die nur unter Beachtung der Copyleft-Beschränkungen genutzt werden darf, und des alternativen Angebots, die Software unter einer anderen Lizenz zu beziehen, was den Lizenznehmer von bestimmten Einschränkungen befreit, und zwar gegen eine Gebühr, die zur Finanzierung der Entwicklung der Software verwendet werden könnte.

Es sollte also deutlich geworden sein, dass die Wahl der Lizenz für Software-Projekte mit viel Bedacht getroffen werden sollte, da davon die Zusammenarbeit mit anderen Projekten, die Kombinierbarkeit mit anderen Komponenten und auch die künftige Ausgestaltung des eigenen Produkts abhängen. Das Copyleft stellt Entwickler dabei vor besondere Herausforderungen.

Open Source Definition and permissive Lizenzen

Auf Seiten der Open-Source-Bewegung ist es die 1998 von Eric S. Raymond und Bruce Perens gegründete *Open Source Inititive* (OSI), die sich maßgeblich mit Fragen der Lizenzierung beschäftigt. Sie hat auch ein standardisiertes Verfahren entwickelt, mit dem sie Software-Lizenzen auf ihre Übereinstimmung mit der von ihr formulierten *Open Source Definition* überprüft. Auf der Website der OSI finden sich aktuell mehr als 80 anerkannte Open-Source-Lizenzen.

Hier führt sie auch Lizenzen als “OSI-approved” auf, die dem Copyleft-Prinzip ausdrücklich widersprechen, allen voran die Gruppe der *BSD-Lizenzen*. Die *Berkeley Software Distribution* (BSD) bezeichnet eine ursprünglich an der Universität Berkeley entwickelte Variante des Betriessystems Unix, aus denen später wiederum freie Projekte wie *NetBSD*, *FreeBSD* und *OpenBSD* hervorgingen. Die diesen Projekten zugrundeliegenden Lizenzen bezeichnet man häufig als *permissive* (“freizügig”). Im Gegensatz zu Copyleft-Lizenzen verfolgen sie gerade *nicht* das Ziel, auch die Nutzungsbedingungen veränderter Varianten festzuschreiben. Das Höchstmaß an Freizügigkeit soll der Software vielmehr zu einer möglichst weiten Verbreitung verhelfen, indem allein den Bearbeitern der Software überlassen wird, wie sie mit den Bearbeitungen verfahren — ob sie sie beispielweise ebenfalls freigeben oder auch als Closed Source behandeln und kommerziell vertreiben.

Wie reduziert eine solch permissive Lizenz sein kann, beweist die *2-Clause BSD License*, auch *Simplified BSD License* oder *FreeBSD License* genannt. Neben der standardisierten Haftungsklausel, mit der sich die Entwickler vor Haftungsansprüchen aus durch die Software verursachten Schäden schützen, besteht die Lizenz lediglich aus den folgenden zwei Regelungen (nicht offizielle Übersetzung):

> Die Weiterverbreitung und Verwendung in Quell- und Binärformen, mit oder ohne Modifikation, ist zulässig, sofern die folgenden Bedingungen erfüllt sind:
>
> 1. Bei der Weitergabe von Quellcode müssen der obige Urheberrechtsvermerk, diese Liste der Bedingungen und der folgende Haftungsausschluss beibehalten werden..
> 2. Weiterverteilungen in binärer Form müssen den obigen Urheberrechtshinweis, diese Liste der Bedingungen und den folgenden Haftungsausschluss in der Dokumentation und/oder anderen Materialien, die mit der Verteilung geliefert werden, wiedergeben.

Creative Commons

Das erfolgreiche Entwicklungskonzept von FLOSS und die damit verbundenen technologischen Fortschritte sorgten dafür, dass man das Open-Source-Prinzip auch auf andere, nicht-technische Bereiche zu übertragen versuchte. Die Aufbereitung und Bereitstellung von Wissen wie auch das kreative Miteinander bei der Lösung komplexer Aufgaben gelten heute als Belege für das erweiterte, inhaltsbezogene Open-Source-Prinzip.

Damit entstand die Notwendigkeit, auch in diesen Bereichen verlässliche Grundlagen zu schaffen, nach denen Arbeitsergebnisse geteilt und bearbeitet werden können. Da die vorliegenden Software-Lizenzen dafür kaum geeignet waren, gab es zahlreiche Versuche, die die spezifischen Anforderungen von wissenschaftlichen Arbeiten bis hin zu digitalisierten Kunstwerken “im Geist von Open Source” in ähnlich griffige Lizenzen überführten.

Die heute mit Abstand wichtigste Initiative dieser Art ist *Creative Commons* (CC), die ihr Anliegen selbst wie folgt zusammenfasst:

> Creative Commons ist eine weltweite Non-Profit-Organisation, die das Teilen und Wiederverwenden von Kreativität und Wissen durch die Bereitstellung freier juristischer Hilfsmittel ermöglicht.
>
> — https://creativecommons.org/faq/#what-is-creative-commons-and-what-do-you-do

Mit Creative Commons geht der Fokus der Rechtevergabe vom Distributor zurück auf den Urheber/Autor. Ein Beispiel: Im klassischen Verlagswesen überträgt ein Urheber meist sämtliche Verwertungsrechte (Druck, Übersetzung etc.) an einen Verlag, der im Gegenzug für die bestmögliche Verbreitung des Werks sorgt. Die deutlich veränderten Distributionswege des Internets versetzen nun den Urheber in die Lage, viele dieser Verwertungsrechte selbst auszuüben und selbst zu entscheiden, wie sein Werk genutzt werden darf. Creative Commons gibt ihm die Möglichkeit, dies einfach und juristisch verlässlich festzulegen, will aber mehr: Der Urheber wird ermutigt, sein Werk als Beitrag einem allgemeinen Prozess des Austauschs und der Zusammenarbeit zur Verfügung zu stellen. Anders als das traditionelle Copyright, das dem

Urheber sämtliche Rechte sichert, die er nach Bedarf an andere überträgt, wählt der Creative-Commons-Ansatz den umgekehrten Weg: Der Urheber stellt sein Werk der Gemeinschaft zur Verfügung, kann aber aus einem Satz von Eigenschaften jene auswählen, die bei der Nutzung des Werks zu beachten sind — je mehr Eigenschaften er auswählt, desto restriktiver die Lizenz.

Und so fragt das "Wähle eine Lizenz"-Prinzip von CC einen Urheber schrittweise die einzelen Eigenschaften ab und generiert daraus die empfohlene Lizenz, die der Urheber zuletzt seinem Werk als Text und Icon zuweisen kann.

Zum besseren Verständnis, hier ein Überblick über die sechs von CC angebotenen Kombinationsmöglichkeiten bzw. Lizenzen:

CC BY ("Namensnennung")

Die freieste Lizenz, nach der jeder das Werk bearbeiten und weitergeben darf, solange er den Urheber nennt.

CC BY-SA ("Namensnennung - Weitergabe unter gleichen Bedingungen")

Wie BY, nur dass das veränderte Werk ausschließlich unter derselben Lizenz weitergegeben werden darf. Das Prinzip erinnert an das Copyleft, da auch hier die Lizenz "vererbt" wird.

CC BY-ND ("Namensnennung-Keine Bearbeitung")

Wie CC BY, nur dass das Werk ausschließlich unverändert/unbearbeitet weitergegeben werden darf.

CC BY-NC ("Namensnennung-Nicht kommerziell")

Das Werk darf unter Nennung des Urhebers zwar bearbeitet und weitergegeben werden, allerings ausschließlich unter nicht-kommerziellen Bedingungen.

CC BY-NC-SA ("Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen")

Wie BY-NC, nur dass das Werk ausschließlich unter denselben Bedingungen weitergegeben werden darf (auch hier also eine Copyleft-ähnliche Lizenz).

CC BY-NC-ND ("Namensnennung - Nicht-kommerziell - Keine Bearbeitung")

Die restriktivste Lizenz: die Weitergabe ist mit Namensnennung des Urhebers erlaubt, allerdings nur unverändert und unter nicht-kommerziellen Bedingungen.

Business-Modelle in Open Source

In der Rückschau wirkt der Siegeszug von FLOSS wie eine Graswurzelbewegung technikbegeisterter Idealisten, die unabhängig von ökonomischen Zwängen und frei von

monetären Abhängigkeiten ihre Arbeit in den Dienst der Allgemeinheit stellten. Gleichzeitig sind im FLOSS-Umfeld milliardenschwere Unternehmen entstanden; stellvertretend sei hier die 1993 gegründete US-amerikanische Firma *Red Hat* mit einem Jahresumsatz von über 3 Milliarden USD (2018) genannt, die 2018 vom IT-Giganten IBM übernommen wurde.

Werfen wir also einen Blick auf das Spannungsverhältnis zwischen der freien und tatsächlich unentgeltlichen Weitergabe hochwertiger Software und den Business-Modellen für ihre Schöpfer, denn eines sollte klar sein: Die zahllosen hochqualifizierten Entwickler freier Software müssen auch Geld verdienen, und das ursprünglich tatsächlich rein nicht-kommerzielle FLOSS-Umfeld muss folglich nachhaltige Business-Modelle entwickeln, um seinen eigenen Kosmos zu bewahren.

Ein häufiger Ansatz, vor allem für größere Projekte in der Anfangsphase, ist das so genannte *Crowdfunding*, also das Einsammeln von Geldspenden über eine Plattform wie *Kickstarter*. Die Spender erhalten dafür von den Entwicklern im Erfolgsfalle, also bei Erreichen zuvor definierter Ziele, eine zuvor festgelegte Gratifikation, seien dies ein unbeschränkter Zugang zum Produkt oder spezielle Features.

Ein anderer Ansatz ist das *Dual Licensing*. Eine freie Software wird parallel unter einer restriktiveren oder gar proprietären Lizenz angeboten, die dem Kunden wiederum weitergehende Services garantiert (Reaktionszeiten bei Fehlern, Updates, Versionen für bestimmte Plattformen o.Ä.). Als ein Beispiel unter vielen sei hier *ownCloud* genannt, das zwar unter der GPL entwickelt wird, dass v.a. Geschäftskunden parallel dazu aber eine “Business Edition” unter einer proprietären Lizenz anbietet.

Nehmen wir ownCloud auch als Beispiel für ein weiteres verbreitetes FLOSS-Business-Modell: Professional Services. Vielen Unternehmen fehlt das notwendige technische Wissen inhouse, um eine komplexe und kritische Software verlässlich und v.a. sicher aufzusetzen und zu betreiben. Darum kaufen sie professionelle Dienstleistungen wie Beratung, Maintenance oder Helpdesk direkt vom Hersteller. Bei dieser Entscheidung spielen auch haftungsrechtliche Fragen eine Rolle, indem die Firma Risiken des Betriebs auf den Hersteller überträgt.

Schafft es eine Software, in ihrem Bereich erfolgreich und beliebt zu werden, sind es periphere Monetarisierungsmöglichkeiten wie Merchandising oder Zertifikate, die Kunden erwerben und damit auf ihren besonderen Status beim Einsatz dieser Software hinzuweisen. So bietet — auch dies nur ein Beispiel unter zahllosen — die Lernplattform *Moodle* die Zertifizierung von Trainern an, die damit ihre Kenntnisse zum Beispiel gegenüber potentiellen Auftraggebern dokumentieren.

Vor allem für webbasierte Technologien ist *Software as Service* ein weiteres Geschäftsmodell. Hier betreibt ein Cloud-Anbieter eine Software wie ein Customer Relationship Management (CRM) oder ein Content Management System (CMS) auf seinen Servern und gewährt seinen Kunden Zugriff auf die installierte Anwendung. Dies erspart dem Kunden Installation und Wartung der Software,

dafür zahlt er für die Nutzung der Software gemäß verschiedenen Parametern, etwa der Anzahl der Nutzer. Verfügbarkeit und Sicherheit spielen als unternehmenskritische Faktoren dabei eine große Rolle.

Zuletzt sei das vor allem bei kleineren Projekten häufige Modell genannt, zu einer freien Sofware per Auftrag kundenspezifische Erweiterungen zu entwickeln. Es obliegt dann meist dem Kunden, wie er mit diesen Erweiterungen verfährt, ob er sie also ebenfalls freigibt oder als Teil seines eigenen Geschäftsmodells unter Verschluss hält.

Eines sollte damit deutlich geworden sein: Obwohl freie Software meist auch unentgeltlich zur Verfügung steht, sind in ihrem Umfeld zahlreiche Business-Modelle entstanden, die von unzähligen Freelancern und Unternehmen weltweit in sehr kreativer Form auch ständig modifiziert und erweitert werden, was zuletzt auch das Fortbestehen der gesamten FLOSS-Bewegung sichert.

Geführte Übungen

1. Wie lauten — in Kurzfrom — die “vier Freiheiten”, wie sie von Richard Stallman und der Free Software Foundation definiert wurden?

Freiheit 0	
Freiheit 1	
Freiheit 2	
Freiheit 3	

2. Wofür steht die Abkürzung FLOSS?

3. Sie haben Software entwickelt und möchten sicherstellen, dass die Software selbst, aber auch alle künftigen darauf aufbauenden Werke ebenso frei bleiben. Welche Lizenz wählen Sie?

CC BY	
GPL v3	
2-Clause BSD License	
LGPL	

4. Welche der folgenden Lizenzen würde man als permissive bezeichnen, welche als Copyleft?

Simplified BSD License	
GPL v3	
CC BY	
CC BY-SA	

5. Sie haben eine Webapplikation geschrieben und unter einer freien Lizenz veröffentlicht. Wie können Sie mit Ihrem Produkt Geld verdienen? Nennen Sie drei Möglichkeiten.

Offene Übungen

1. Unter welcher Lizenz (einschließlich Version) stehen die folgenden Anwendungen?

Apache HTTP Server	
MySQL Community Server	
Wikipedia articles	
Mozilla Firefox	
GIMP	

2. Sie möchten Ihre Software unter der GNU GPL v3 veröffentlichen. Welche Schritte sollten Sie beachten?

3. Sie haben proprietäre Software geschrieben und würden sie gerne mit freier Software unter der GPL v3 kombinieren. Dürfen Sie das bzw. was müssen Sie beachten?

4. Warum hat die Free Software Foundation als Ergänzung zur GNU GPL die *GNU Affero General Public License 3* (GNU AGPL) veröffentlicht?

5. Nennen Sie drei Beispiele freier Software, die auch als “Business Edition”, also in einer kostenpflichtigen Version angeboten werden.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Gemeinsamkeiten und Unterschiede zwischen freier und Open Source Software (FLOSS)
- FLOSS-Lizenzen, deren Wichtigkeit und Probleme
- Copyleft vs. permissive Lizenzen
- FLOSS-Geschäftsmodelle

Antworten zu den geführten Übungen

1. Wie lauten — in Kurzfrom — die “vier Freiheiten”, wie sie von Richard Stallman und der Free Software Foundation definiert wurden?

Freiheit 0	die Software ausführen
Freiheit 1	die Software untersuchen und anpassen (Source Code)
Freiheit 2	die Software redistribuieren
Freiheit 3	die Sofware verbessern und die Verbesserungen freigeben

2. Wofür steht die Abkürzung FLOSS?

   Free/Libre Open Source Software

3. Sie haben Software entwickelt und möchten sicherstellen, dass die Software selbst, aber auch alle künftigen darauf aufbauenden Ergebnisse ebenso frei bleiben. Welche Lizenz wählen Sie?

CC BY	
GPL v3	X
2-Clause BSD License	
LGPL	

4. Welche der folgenden Lizenzen würde man als permissive bezeichnen, welche als Copyleft?

Simplified BSD License	permissive
GPL v3	Copyleft
CC BY	permissive
CC BY-SA	Copyleft

5. Sie haben eine Webapplikation geschrieben und unter einer freien Lizenz veröffentlicht. Wie können Sie mit Ihrem Produkt Geld verdienen? Nennen Sie drei Möglichkeiten.
   - Dual licensing, also das Anbieten einer kostenpflichtigen “Business Edition”
   - Anieten von Hosting, Service und Support
   - Entwickeln von Erweiterungen für den Kundens

Antworten zu den offenen Übungen

1. Unter welcher Lizenz (einschließlich Version) stehen die folgenden Anwendungen?

Apache HTTP Server	Apache License 2.0
MySQL Community Server	GPL 2
Wikipedia articles (English)	Creative Commons Attribution Share-Alike license (CC-BY-SA)
Mozilla Firefox	Mozilla Public License 2.0
GIMP	GPL 3

2. Sie möchten Ihre Software unter der GNU GPL v3 veröffentlichen. Welche Schritte sollten Sie beachten?
   - Sichern Sie sich bei Bedarf zum Beispiel gegenüber dem Arbeitgeber mit einem Copyright-Verzicht ab, dass Sie die Lizenz festlegen können.
   - Versehen Sie jede Datei der Software mit einem Copyright-Vermerk.
   - Fügen Sie eine Datein namens `COPYING` mit dem vollständigen Lizenztext zu Ihrer Software hinzu.
   - Ergänzen Sie einen Hinweis auf die Lizenz in jeder Datei.
3. Sie haben proprietäre Software geschrieben und würden sie gerne mit freier Software unter der GPL v3 kombinieren. Dürfen Sie das bzw. was müssen Sie beachten?

   Die FAQ der FSF geben hier Auskunft: Sofern Ihre proprietäre Software und die freie Software voneinander getrennt bleiben, ist die Kombination möglich. Sie müssen allerdings sicherstellen, dass diese Trennung technisch gewährleistet ist und für die Benutzer erkennbar ist. Wenn Sie die freie Software so integrieren, dass sie Teil ihres Produkts wird, müssen Sie das Produkt nach dem Prinzip des Copyleft ebenfalls unter der GPL veröffentlichen.

4. Warum hat die Free Software Foundation als Ergänzung zur GNU GPL die *GNU Affero General Public License 3* (GNU AGPL) veröffentlicht?

   Die GNU AGPL schließt eine Lizenz-Lücke, die insbesondere bei freier Software entsteht, die auf einem Server gehostet wird: Nimmt ein Entwickler Änderungen an der Software vor, ist er gemäß der GPL nicht verpflichtet, diese Änderungen zugänglich zu machen, da er zwar den Zugriff auf das Programm erlaubt, aber das Programm nicht im GPL-Sinne weitergibt. Die GNU AGPL hingegen schreibt vor, dass die Software mit sämtlichen Änderungen zum Download angeboten werden muss.

5. Nennen Sie drei Beispiele für freie Software, die auch in einer “Business Edition” angeboten wird, z.B. als kostenpflichtige Version.

   MySQL, Zammad, Nextcloud

1.4 IKT-Fähigkeiten und Arbeiten mit Linux

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 1.4

Gewichtung

2

Hauptwissensgebiete

- Desktop-Fähigkeiten
- Zugang zur Befehlszeile
- Linux, Cloud-Computing und Virtualisierung in der Industrie

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- Verwendung eines Browsers, Beachtung des Datenschutzes, Konfigurationsoptionen, Suche im Internet und Speichern von Inhalten
- Terminal und Konsole
- Passwörter
- Privatsphäre-Einstellungen und -Tools
- Verwendung gängiger Open-Source-Anwendungen in Präsentationen und Projekten

1.4 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	1 Die Linux-Community und Karriere im Open-Source-Umfeld
Lernziel:	1.4 IKT-Fähigkeiten und Arbeiten mit Linux
Lektion:	1 von 1

Einführung

Es gab eine Zeit, in der die Arbeit mit Linux auf dem Desktop als schwierig galt, da dem System viele ausgefeiltere Desktop-Anwendungen und Konfigurationstools fehlten, die andere Betriebssysteme hatten. Das lag unter anderem daran, dass Linux deutlich jünger war als viele andere Betriebssysteme und es einfacher war, zunächst wichtigere Kommandozeilenanwendungen zu entwickeln und die komplexeren grafischen Tools für später zu belassen. Da Linux ursprünglich für fortgeschrittenere Benutzer gedacht war, galt das auch nicht als Problem. Aber diese Zeiten sind längst vorbei. Linux-Desktop-Umgebungen sind heute sehr ausgereift und lassen in Bezug auf Funktionalität und Benutzerfreundlichkeit keine Wünsche offen. Dennoch gilt die Befehlszeile immer noch als ein mächtiges Werkzeug, das fortgeschrittene Anwender täglich nutzen. In dieser Lektion werfen wir einen Blick auf einige der grundlegenden Desktop-Fähigkeiten, die Sie benötigen, um das beste Werkzeug für den richtigen Job auszuwählen, einschließlich der Befehlszeile.

Linux-Benutzeroberflächen

Bei der Arbeit mit einem Linux-System interagieren Sie entweder über eine Befehlszeile oder über eine grafische Benutzeroberfläche. Beide Wege ermöglichen den Zugriff auf zahlreiche Anwendungen, die die Ausführung fast aller Aufgaben mit dem Computer unterstützen. Während Sie in einer früheren Lektion bereits eine Reihe häufig verwendeter Anwendungen kennengelernt haben, beginnen wir diese Lektion mit einem genaueren Blick auf Desktop-Umgebungen, Zugriffsmöglichkeiten auf das Terminal und Tools für Präsentationen und Projektmanagement.

Desktop-Umgebungen

Linux hat einen modularen Ansatz, bei dem verschiedene Teile des Systems von verschiedenen Projekten und Entwicklern entwickelt werden, von denen jeder eine bestimmte Anforderung oder Zielsetzung erfüllt. Darum stehen auch mehrere Desktop-Umgebungen zur Auswahl. Gemeinsam mit Paketmanagern ist die Standard-Desktop-Umgebung einer der Hauptunterschiede zwischen den vielen Distributionen. Im Gegensatz zu proprietären Betriebssystemen wie Windows und macOS, bei denen die Benutzer auf die mit ihrem Betriebssystem mitgelieferte Desktop-Umgebung beschränkt sind, gibt es die Möglichkeit, mehrere Umgebungen zu installieren und diejenige auszuwählen, die sich am besten an Sie und Ihre Bedürfnisse anpasst.

Grundsätzlich gibt es zwei große Desktop-Umgebungen in der Linux-Welt: *Gnome* und *KDE*. Beide sind sehr vollständig, haben jeweils eine große Community hinter sich und verfolgen dasselbe Ziel, wenn auch mit leicht unterschiedlichen Ansätzen. Kurz gesagt, folgt Gnome dem KISS (“keep it simple stupid”) Prinzip, mit sehr schlanken und sauberen Anwendungen. Demgegenüber hat KDE eine andere Perspektive mit einer größeren Auswahl an Anwendungen und der Möglichkeit für den Benutzer, jede Konfigurationseinstellung in der Umgebung zu ändern.

Während Gnome-Anwendungen auf dem GTK-Toolkit (geschrieben in C) basieren, nutzen KDE-Anwendungen die Qt-Bibliothek (geschrieben in C++). Einer der Hauptgründe, Anwendungen mit demselben grafischen Toolkit zu entwickeln, besteht darin, dass die Anwendungen ein ähnliches Look-and-Feel teilen, was wiederum dem Benutzer den Eindruck von Einheitlichkeit vermittelt. Ein weiteres wichtiges Merkmal ist, dass die Verwendung einer gemeinsamen grafischen Bibliothek für viele häufig verwendete Anwendungen Speicherplatz spart und zugleich die Ladezeit verkürzt, sobald die Bibliothek zum ersten Mal geladen wurde.

Zur Kommandozeile gelangen

Für uns ist eine der wichtigsten Anwendungen der grafische Terminalemulator. Die Bezeichnung erklärt sich daher, dass er in einer grafischen Umgebung die alten seriellen Terminals (oft Fernschreiber) emuliert. Tatsächlich waren dies Clients, die mit einer entfernten Maschine verbundene waren, auf der die eigentlichen Berechnungen stattfanden. Das waren wirklich

einfache Computer ohne jede Grafik, auf denen die ersten Versionen von Unix verwendet wurden.

In Gnome heißt eine solche Anwendung *Gnome Terminal*, während sie in KDE als *Konsole* zu finden ist. Aber es stehen viele weitere zur Wahl, wie z.B. *Xterm*. Diese Anwendungen sind für uns der Zugang zu einer Kommandozeilenumgebung, um mit einer Shell interagieren zu können.

Schauen Sie sich also das Anwendungsmenü Ihrer bevorzugten Distribution für eine Terminalanwendung an. Bei allen Unterschieden zwischen diesen Anwendungen bietet jede das, was notwendig ist, um Vertrauen im Umgang mit der Kommandozeile zu gewinnen.

Eine weitere Möglichkeit, in das Terminal zu gelangen, ist das virtuelle TTY. Sie erreichen es über die Tastenkombination `Strg` + `Alt` + `F#`, wobei `F#` für eine der Funktionstasten von 1 bis 7 steht. Wahrscheinlich können einige der ersten Kombinationen Ihren Sitzungsmanager oder Ihre grafische Umgebung ausführen. Die anderen zeigen eine Eingabeaufforderung an, in der sie nach Ihrem Login-Namen fragen:

```
Ubuntu 18.10 arrelia tty3
arrelia login:
```

`arrelia` ist in diesem Fall der Hostname der Maschine, und `tty3` ist das Terminal, das nach Verwendung der obigen Tastenkombination verfügbar ist, plus die Taste `F3`, also `Ctrl` + `Alt` + `F3`.

Nachdem Sie Ihren Benutzernamen und Ihr Passwort eingegeben haben, gelangen Sie schließlich in ein Terminal, aber es gibt hier keine grafische Umgebung, so dass Sie nicht in der Lage sein werden, die Maus zu benutzen oder grafische Anwendungen auszuführen, ohne zuerst eine X- oder Wayland-Sitzung zu starten. Aber das geht über das Thema dieser Lektion hinaus.

Präsentationen und Projekte

Das wichtigste Werkzeug für Präsentationen unter Linux ist *LibreOffice Impress*. Es ist Teil der Open-Source-Office-Pakets namens *LibreOffice*. Denken Sie an LibreOffice als Open-Source-Ersatz für das gleichwertige *Microsoft Office*. Es kann sogar dessen PPT- und PPTX-Dateien öffnen und speichern, die nativ zu *Powerpoint* gehören. Trotzdem ist dringend empfohlen, das native ODP Impress-Format zu verwenden. ODP ist Teil des größeren *Open Document Format*, einem internationalen Standard für diese Art von Datei. Dies ist besonders wichtig, wenn Sie Ihre Dokumente über viele Jahre hinweg zugänglich halten und sich weniger um Kompatibilitätsprobleme kümmern wollen. Da es sich um einen offenen Standard handelt, ist es für jeden möglich, das Format ohne Lizenzgebühren oder Lizenzen zu implementieren. Dies macht es Ihnen auch möglich, andere Präsentationssoftware auszuprobieren, die Ihnen vielleicht besser gefällt, und Ihre Dateien mitzunehmen, da es sehr wahrscheinlich ist, dass sie mit diesen

neueren Softwareprodukten kompatibel sind.

Aber wenn Sie Code den grafischen Oberflächen vorziehen, gibt es ein paar Werkzeuge zur Auswahl: *Beamer* ist eine *LaTeX*-Klasse, die Folienpräsentationen aus LaTeX-Code erstellt. LaTeX selbst ist ein Satzsystem, das hauptsächlich zum Schreiben wissenschaftlicher Dokumente verwendet wird, insbesondere wegen seiner Fähigkeit, komplexe mathematische Formeln zu verarbeiten, mit denen andere Softwareprogramme Schwierigkeiten haben. Wenn Sie an der Universität sind und sich mit Gleichungen und anderen mathematischen Problemen befassen müssen, kann Beamer Ihnen eine Menge Zeit sparen.

Eine weitere Option ist *Reveal.js*, ein großartiges NPM-Paket (NPM ist der Standard-Paketmanager von NodeJS), mit dem Sie schöne Präsentationen über das Internet erstellen können. Wenn Sie also HTML und CSS schreiben können, wird Reveal.js die meisten JavaScript-Funktionen mitbringen, die für die Erstellung schöner und interaktiver Präsentationen erforderlich sind, die sich gut an jede Auflösung und Bildschirmgröße anpassen.

Wenn Sie einen Ersatz für *Microsoft Project* suchen, probieren Sie *GanttProject* oder *ProjectLibre* aus. Beide sind ihrem proprietären Gegenstück sehr ähnlich und mit Projektdateien kompatibel.

Industrielle Linux-Anwendungen

Linux spielt eine große Rolle in der Software- und Internetbranche. Websites wie W3Techs berichten, dass etwa 68% der Webserver im Internet unter Unix laufen und der allergrößte Teil davon unter Linux.

Diese große Akzeptanz ist nicht nur auf die freie Natur von Linux (sowohl im Sinne von Freibier als auch im Sinne von Meinungsfreiheit) zurückzuführen, sondern auch auf seine Stabilität, Flexibilität und Leistung. Sie machen es Anbietern möglich, ihre Dienste kostengünstiger und besser skalierbar anzubieten. Ein bedeutender Teil der Linux-Systeme läuft heute in der Cloud, also in einem IaaS- (Infrastructure as a Service), PaaS- (Platform as a Service) oder SaaS- (Software as a Service) Modell.

IaaS ist eine Möglichkeit, die Ressourcen eines großen Servers zu verteilen, indem man ihnen Zugang zu virtuellen Maschinen bietet, die tatsächlich mehrere Betriebssysteme sind, die als Gäste auf einer Hostmaschine laufen, und zwar über eine wichtige Software, die als *Hypervisor* bezeichnet wird. Der Hypervisor ist dafür verantwortlich, dass diese Gastbetriebssysteme ausgeführt werden können, indem er die auf der Hostmaschine verfügbaren Ressourcen für diese Gäste trennt und verwaltet. Das nennen wir *Virtualisierung*. Im IaaS-Modell bezahlen Sie nur für den Teil der Ressourcen, den Ihre Infrastruktur nutzt.

Linux hat drei bekannte Open-Source-Hypervisoren: *Xen*, *KVM* und *VirtualBox*. Xen ist

wahrscheinlich der älteste von ihnen. KVM überholte Xen als prominentester Linux-Hypervisor. RedHat hat seine Entwicklung unterstützt und nutzte es — ebenso wie andere Anbieter — in Public-Cloud-Diensten wie auch in privaten Cloud-Setups verwendet. VirtualBox gehört seit der Übernahme von Sun Microsystems zu Oracle und wird in der Regel von Endbenutzern wegen seiner einfachen Bedienung und Administration eingesetzt.

PaaS und SaaS hingegen bauen technisch und konzeptionell auf dem IaaS-Modell auf: In PaaS haben die Anwender statt einer virtuellen Maschine Zugriff auf eine Plattform, auf der sie ihre Anwendungen deployen und ausführen, um den Aufwand für Systemadministrationsaufgaben und Betriebssystem-Updates zu verringern. Heroku ist ein gängiges PaaS-Beispiel, bei dem Programmcode nur ohne Pflege der zugrunde liegenden Container und virtuellen Maschinen ausgeführt werden kann.

Schließlich ist SaaS das Modell, bei dem Sie normalerweise für ein Abonnement bezahlen, um eine Software zu nutzen, ohne sich um etwas anderes zu kümmern. *Dropbox* und *Salesforce* sind zwei gute Beispiele für SaaS. Die meisten dieser Dienste werden über einen Webbrowser aufgerufen.

Ein Projekt wie *OpenStack* ist eine Sammlung von Open-Source-Software, die verschiedene Hypervisoren und andere Tools nutzt, um eine komplette IaaS-Cloud-Umgebung anzubieten, indem sie die Leistungsfähigkeit von Computer-Clustern in ihrem eigenen Rechenzentrum nutzt. Der Aufbau einer solchen Infrastruktur ist jedoch nicht trivial.

Datenschutzprobleme bei der Nutzung des Internets

Der Webbrowser ist heute eine grundlegende Software auf jedem Desktop, aber vielen Anwendern fehlt das Wissen, ihn sicher zu nutzen. Während immer mehr Dienste über einen Webbrowser aufgerufen werden, werden fast alle Aktionen, die über einen Browser ausgeführt werden, von verschiedenen Stellen verfolgt und analysiert. Die Sicherung des Zugangs zu Internetdiensten und die Verhinderung von Tracking ist ein wichtiger Aspekt der sicheren Nutzung des Internets.

Cookie Tracking

Nehmen wir an, Sie haben eine E-Commerce-Website besucht, ein Produkt ausgewählt und es in den Warenkorb gelegt. In letzter Sekunde haben Sie sich jedoch entschieden, es sich noch einmal zu überlegen. Nach einer Weile sehen Sie Anzeigen desselben Produkts, die Ihnen im Internet folgen. Wenn Sie auf die Anzeigen klicken, werden Sie sofort wieder auf die Produktseite dieses Geschäfts geschickt. Es ist nicht ungewöhnlich, dass die Produkte, die Sie in den Warenkorb gelegt haben, noch da sind und nur darauf warten, dass Sie sich entscheiden, sie zu kaufen. Haben Sie sich jemals gefragt, wie das geht? Wie erscheint die passende Anzeige auf einer ganz anderen

Webseite? Die Antwort auf diese Fragen heißt *Cookie Tracking*.

Cookies sind kleine Dateien, die eine Website auf Ihrem Computer ablegen kann, um Informationen zu speichern und abzurufen, die für Ihre Navigation nützlich sein können. Sie werden seit vielen Jahren verwendet und sind eine der ältesten Möglichkeiten, Daten auf Kundenseite zu speichern. Ein gutes Beispiel für ihre Verwendung sind eindeutige Einkaufskarten-IDs, so dass der Shop, wenn Sie in ein paar Tagen wieder auf die gleiche Website zurückkehren, Sie an die Produkte erinnern kann, die Sie bei Ihrem letzten Besuch in Ihren Warenkorb gelegt haben. So sparen Sie Zeit bei der erneuten Suche nach diesen Produkten.

Das ist in der Regel in Ordnung, da die Website Ihnen ein nützliches Feature bietet und keine Daten an Dritte weitergibt. Aber was ist mit den Anzeigen, die erscheinen, während Sie auf anderen Webseiten surfen? Hier kommen die Werbenetzwerke ins Spiel. Werbenetzwerke sind Unternehmen, die Anzeigen für E-Commerce-Sites wie die in unserem Beispiel auf der einen Seite und Monetarisierung für Websites auf der anderen Seite anbieten. Content-Ersteller wie z.B. Blogger können etwas Platz für diese Werbenetzwerke auf ihrem Blog zur Verfügung stellen, gegen eine Provision für den Umsatz, der durch diese Werbung generiert wird.

Aber woher wissen sie, welches Produkt sie Ihnen zeigen sollen? Indem sie in der Regel auch ein Cookie aus dem Werbenetzwerk speichern, das sie bei Ihrem Besuch oder bei der Suche nach einem bestimmten Produkt auf der E-Commerce-Website gespeichert haben. So ist das Netzwerk in der Lage, Informationen über dieses Cookie überall dort abzurufen, wo das Netzwerk Werbung hat, und die Korrelation mit den Produkten herzustellen, die Sie interessiert haben. Das ist eine der häufigsten Wege, jemanden über das Internet zu verfolgen. Das oben genannte Beispiel nutzt E-Commerce, um die Dinge greifbarer zu machen, aber Social-Media-Plattformen tun dasselbe mit ihren “Like”- oder “Share”-Buttons und ihrem sozialen Login.

Eine Möglichkeit, das zu vermeiden, besteht darin, Websites von Drittanbietern das Speichern von Cookies in Ihrem Browser zu verbieten. Auf diese Weise kann nur die von Ihnen besuchte Website ihre Cookies speichern. Beachten Sie jedoch, dass einige “legitime” Funktionen dann möglicherweise nicht mehr gut funktionieren, da viele Websites heute auf Dienste von Drittanbietern angewiesen sind. Suchen Sie im Add-On-Repository Ihres Browsers nach einem Cookie-Manager, um genaue Kontrolle darüber zu haben, welche Cookies auf Ihrem Computer gespeichert werden.

Do Not Track (DNT)

Ein weiterer häufiger Irrtum betrifft eine bestimmte Browserkonfiguration, bekannt als “Do Not Track” oder kurz DNT. Es kann grundsätzlich in jedem aktuellen Browser eingeschaltet werden. Ähnlich wie beim privaten Modus glauben sehr viele, sie könnten mit dieser Konfiguration nicht verfolgt werden. Leider ist dem nicht immer so. Derzeit ist DNT nur eine Möglichkeit für Sie, den

von Ihnen besuchten Websites mitzuteilen, dass Sie von ihnen nicht verfolgt werden möchten. Aber tatsächlich entscheiden immer noch die Websites, ob sie Ihren Wunsch respektieren oder nicht. Mit anderen Worten, mit DNT können Sie sich vom Website-Tracking abwählen, aber es gibt keine Garantie für diese Wahl.

Technisch gesehen geschieht dies einfach durch Senden eines zusätzlichen Flags im Header des HTTP-Request-Protokolls (`DNT: 1`) bei der Anforderung von Daten von einem Webserver. Wenn Sie mehr über dieses Thema erfahren möchten, ist die Website https://allaboutdnt.com ein guter Ausgangspunkt.

“Private” Fenster

Sicher haben Sie die Anführungszeichen in der Überschrift bemerkt, denn diese Fenster sind nicht so privat, wie die viele denken. Die Namen können variieren, aber sie heißen “Privatmodus”, “Inkognito” oder “Anonym”, je nach Browser, den Sie verwenden.

In Firefox nutzen Sie es ganz einfach, indem Sie die Tastenkombination `Strg` + `Shift` + `P` drücken. In Chrome drücken Sie einfach `Strg` + `Shift` + `N`. Tatsächlich wird dadurch eine neue Sitzung geöffnet, die normalerweise keine Konfiguration oder Daten aus Ihrem Standardprofil enthält. Wenn Sie das private Fenster schließen, löscht der Browser automatisch alle von dieser Sitzung erzeugten Daten und hinterlässt keine Spuren auf dem verwendeten Computer. Das bedeutet, dass auf diesem Computer keine personenbezogenen Daten wie Verlauf, Passwörter oder Cookies gespeichert werden.

So missverstehen viele Menschen dieses Konzept als anonymes Surfen im Internet, was es aber eben nicht ganz ist. Eine Sache, die der Datenschutz- oder Inkognito-Modus tut, ist, das so genannte Cookie-Tracking zu vermeiden: Wenn Sie eine Website besuchen, kann sie eine kleine Datei auf Ihrem Computer speichern, die eine ID enthält, die zur Nachverfolgung verwendet werden kann, es sei denn, Sie konfigurieren Ihren Browser so, dass er keine Cookies von Drittanbietern akzeptiert, können Werbenetzwerke oder andere Unternehmen diese ID speichern und abrufen und Ihr Surfen auf Websites tatsächlich verfolgen.

Darüber hinaus gibt es für Websites und andere Peers im Internet viele andere Techniken, um Sie zu verfolgen. Der private Modus bringt Ihnen also ein gewisses Maß an Anonymität, ist aber nur auf dem Computer, den Sie verwenden, völlig privat. Wenn Sie von einem öffentlichen Computer aus auf Ihr E-Mail-Konto oder Ihre Bank-Website zugreifen, etwa von einem Flughafen oder einem Hotel aus, sollten Sie auf jeden Fall den privaten Modus Ihres Browsers nutzen. In anderen Situationen kann es Vorteile geben, aber Sie sollten genau wissen, welche Risiken Sie vermeiden und welche haben keine Auswirkungen. Wenn Sie einen öffentlich zugänglichen Computer verwenden, sollten Sie sich darüber im Klaren sein, dass andere Sicherheitsbedrohungen wie Malware oder Key Logger bestehen können. Seien Sie vorsichtig, wenn Sie personenbezogene

Daten, einschließlich Benutzernamen und Passwörter, auf solchen Computern eingeben oder wenn Sie vertrauliche Daten herunterladen oder kopieren.

Das richtige Passwort wählen

Eine der schwierigsten Situationen für jeden Benutzer ist die Wahl eines sicheren Passworts für die von ihm genutzten Dienste. Sie haben sicherlich schon einmal gehört, dass Sie weder gängige Kombinationen wie `qwerty`, `123456` oder `654321` noch leicht zu erratende Zahlen wie Ihren Geburtstag (oder den eines Verwandten) oder Ihre Postleitzahl verwenden sollten, denn das sind alles sehr naheliegende Kombinationen und die ersten Versuche, die ein Eindringling unternehmen wird, um Zugang zu Ihrem Konto zu erhalten.

Es gibt bekannte Techniken zur Erstellung eines sicheren Passworts. Eine der bekanntesten ist die Zusammenstellung eines Satzes, der Sie an diesen Dienst erinnert und die Anfangsbuchstaben jedes Wortes aufnimmt. Nehmen wir an, Sie möchten beispielsweise ein gutes Passwort für Facebook erstellen. In diesem Fall könnten Sie sich einen Satz wie “Ich wäre glücklich, wenn ich 1000 Freunde wie Mike hätte” ausdenken. Wählen Sie den ersten Buchstaben jedes Wortes und das endgültige Passwort lautet `Iwgwi1000FwMh`. Dies würde zu einem 13-stelligen Passwort führen, das lang genug ist, um schwer zu erraten und gleichzeitig leicht zu merken ist (solange Sie sich an den Satz und den “Algorithmus” zum Abrufen des Passworts erinnern).

Sätze sind in der Regel leichter zu merken als Passwörter, aber auch diese Methode hat ihre Grenzen. Wir müssen heutzutage Passwörter für so viele Dienste erstellen, und da wir sie unterschiedlich häufig verwenden, wird es irgendwann sehr schwierig, sich an alle Sätze zu dem Zeitpunkt zu erinnern, an dem wir sie brauchen. Was können wir also tun? Man könnte auf die Idee verfallen, ein paar gute Passwörter zu erstellen und sie bei ähnlichen Diensten wiederzuverwenden.

Leider ist das keine gute Idee. Sie haben wahrscheinlich gehört, dass Sie dasselbe Passwort nicht für verschiedene Dienste verwenden sollten, denn das Problem ist, dass ein Dienst Ihr Passwort durchsickern lassen kann (ja, das passiert immer wieder), und irgendjemand, der Zugang dazu hat, wird versuchen, dieselbe E-Mail- und Passwortkombination bei anderen beliebten Diensten im Internet auszuprobieren, in der Hoffnung, dass Sie genau das getan haben: Passwörter wiederverwendet. Und wenn das der Fall ist, werden Sie am Ende nicht nur mit einem Dienst Probleme haben, sondern mit mehreren. Und wie immer ist es so: Wir glauben es nicht, bis es uns passiert.

Was also können wir tun, um uns zu schützen? Einer der heute sichersten Ansätze ist die Verwendung eines so genannten *Passwortmanagers*. Passwortmanager sind eine Software, die im Wesentlichen alle Ihre Passwörter und Benutzernamen in einem verschlüsselten Format speichert, das mit einem Master-Passwort entschlüsselt werden kann, so dass Sie sich nur ein

gutes Passwort merken müssen, da der Manager alle anderen für Sie sicher verwahrt.

KeePass ist einer der bekanntesten und funktionsreichsten Open-Source-Passwortmanager. Es speichert Ihre Passwörter in einer verschlüsselten Datei in Ihrem Dateisystem. Dass es sich um Open-Source-Software handelt, ist in diesem Zusammenhang wichtig, weil das garantiert, dass Ihre Daten nicht anderweitig genutzt werden, denn jeder Entwickler kann den Code überprüfen und weiß genau, wie er funktioniert. Dieses Maß an Transparenz ist mit proprietärem Code nicht zu erreichen. KeePass steht für die meisten Betriebssysteme zur Verfügung, einschließlich Windows, Linux und macOS — ebenso wie für mobile Systeme wie iOS und Android. Es umfasst auch ein Plugin-System, das seine Funktionalität weit über die Standardeinstellungen hinaus erweitert.

Bitwarden ist eine weitere Open-Source-Lösung mit einem ähnlichen Ansatz, aber statt Ihre Daten lokal in einer Datei zu speichern nutzt es einen Cloud-Server. So halten Sie alle Ihre Geräte synchron greifen über das Internet leichter auf Ihre Passwörter zu. Bitwarden ist eines der wenigen Projekte, das nicht nur die Clients, sondern auch den Cloud-Server als Open-Source-Software zur Verfügung stellt. Sie können also Ihre eigene Version von Bitwarden hosten und sie anderen zur Verfügung stellen, etwa Ihrer Familie oder den Mitarbeitern Ihres Unternehmens. Dies gibt Ihnen Flexibilität, aber auch die volle Kontrolle darüber, wie ihre Passwörter gespeichert und verwendet werden.

Eines der wichtigsten Dinge, die Sie bei der Verwendung eines Passwortmanagers beachten sollten, ist die Erstellung eines zufälligen Passworts für jeden einzelnen Dienst, da Sie sich die Passwörter ohnehin nicht mehr merken müssen. Es wäre nutzlos, einen Passwortmanager zu verwenden, um recycelte oder leicht zu erratende Passwörter zu speichern. Darum bieten Ihnen die meisten einen Zufallspasswortgenerator an, mit dem Sie diese erstellen.

Verschlüsselung

Wann immer Daten übertragen oder gespeichert werden, sind Vorkehrungen zu treffen, damit Dritte nicht darauf zugreifen können. Über das Internet übertragene Daten passieren eine Reihe von Routern und Netzwerken, in denen Dritte auf den Netzwerkverkehr zugreifen können. Ebenso lassen sich auf physischen Datenträgern gespeicherte Daten von jedem lesen, der in den Besitz dieser Datenträger gelangt. Zur Vermeidung eines solchen Zugriffs sollten vertrauliche Informationen verschlüsselt werden, bevor sie einen Rechner verlassen.

TLS

Transport Layer Security (TLS) ist ein Protokoll, das Sicherheit von Netzwerkverbindungen durch den Einsatz von Kryptographie bietet. TLS ist der Nachfolger von *Secure Sockets Layer* (SSL), das wegen schwerwiegender Fehler nicht mehr eingesetzt wird. TLS wurde immer wieder

überarbeitet, um sich anzupassen und sicherer zu werden, und liegt aktuell in Version 1.3 vor. Es bietet durch so genannte symmetrische und Public-Key-Kryptographie sowohl Vertraulichkeit als auch Authentizität. Das bedeutet, dass niemand Ihre Kommunikation mit diesem Server während dieser Sitzung abhören oder verändern kann.

Die wichtigste Aufgabe besteht darin herauszufinden, ob eine Website vertrauenswürdig ist. Sie sollten nach dem Symbol “Sperren” in der Adressleiste des Browsers suchen, auf das Sie bei Bedarf klicken, um das Zertifikat zu überprüfen, das eine wichtige Rolle im HTTPS-Protokoll spielt.

TLS ist das, was im HTTPS-Protokoll (*HTTP über TLS*) verwendet wird, um den Versand sensibler Daten (zum Beispiel Ihre Kreditkartennummer) über das Internet zu gewährleisten. Die Erklärung, wie TLS funktioniert, geht weit über das Thema dieser Lektion hinaus, aber weitere Informationen finden Sie auf Wikipedia und im Mozilla wiki.

Datei- und E-Mail-Verschlüsselung mit GnuPG

Es gibt viele Tools zur Absicherung von E-Mails, aber eines der wichtigsten ist sicherlich *GnuPG*. GnuPG steht für *GNU Privacy Guard* und ist eine Open-Source-Implementierung von *OpenPGP*, einem internationalen Standard, der in RFC 4880 kodifiziert ist.

GnuPG dient dazu, Texte, E-Mails, Dateien, Verzeichnisse und sogar ganze Festplattenpartitionen zu signieren, zu verschlüsseln und zu entschlüsseln. Es nutzt Public-Key-Kryptographie und ist weit verbreitet. Kurz zusammengefasst, GnuPG erstellt Paar von Dateien mit Ihrem öffentlichen und Ihrem privaten Schlüssel. Wie der Name schon sagt, ist der öffentliche Schlüssel jedermann zugänglich, während der private Schlüssel geheim zu halten ist. Andere verwenden Ihren öffentlichen Schlüssel, um Daten zu verschlüsseln, die nur Ihr privater Schlüssel entschlüsseln kann.

Sie nutzen Ihren privaten Schlüssel, um eine Datei oder E-Mail zu signieren, die mit dem passenden öffentlichen Schlüssel validiert werden kann. Diese digitale Signatur funktioniert analog zur realen Signatur: Solange Sie der Einzige sind, der Zugriff auf Ihren privaten Schlüssel hat, kann der Empfänger sicher sein, dass Sie tatsächlich selbst unterschrieben haben. Durch kryptographische Hash-Funktionalität garantiert GnuPG, dass nach der Signatur keine Änderungen am Inhalt vorgenommen wurden, da diese die Signatur ungültig machen würden.

GnuPG ist ein sehr mächtiges und in gewissem Maße auch komplexes Werkzeug. Weitere Informationen finden Sie auf der Projekt-Website und im Archlinux-Wiki (Archlinux-Wiki ist eine sehr gute Informationsquelle, auch wenn Sie Archlinux nicht verwenden).

Festplattenverschlüsselung

Eine gute Möglichkeit, Ihre Daten zu schützen, besteht in der Verschlüsselung Ihrer gesamten Festplatte oder Partition. Es gibt dafür viele Open-Source-Lösungen. Sie unterscheiden sich zum Teil erheblich in der Funktionsweise und im Grad der Verschlüsselung. Grundsätzlich gibt es zwei Methoden: *Stacked*- und *Block*-Device-Verschlüsselung.

Stacked-Dateisystemlösungen werden auf dem bestehenden Dateisystem implementiert. Hier werden Dateien und Verzeichnisse verschlüsselt, bevor sie auf dem Dateisystem gespeichert, und entschlüsselt, nachdem sie gelesen werden. Die Dateien liegen also auf dem Host-Dateisystem in verschlüsselter Form (ihr Inhalt und in der Regel auch ihre Datei-/Ordnernamen werden durch Zufallsdaten ersetzt), aber sie existieren noch in diesem Dateisystem, wie sie ohne Verschlüsselung als normale Dateien, Symlinks, Hardlinks, etc. dort liegen.

Demgegenüber erfolgt die Verschlüsselung von Block Devices unterhalb der Dateisystemebene und stellt sicher, dass alles, was auf ein Block Device geschrieben wird, verschlüsselt wird. Wenn Sie sich den Block ansehen, während er offline ist, sieht er wie ein Haufen von Zufallsdaten aus und Sie können nicht einmal den Dateisystemtyp erkennen, ohne zuvor zu entschlüsseln. Sie können also nicht sagen, was eine Datei oder ein Verzeichnis ist, wie groß sie ist und um welche Art von Daten es sich handelt, denn Metadaten, Verzeichnisstruktur und Berechtigungen sind ebenfalls verschlüsselt.

Beide Methoden haben ihre Vor- und Nachteile. Werfen Sie vor allem einen Blick auf *dm-crypt*, den De-facto-Standard für Blockverschlüsselung für Linux-Systeme, da es nativ im Kernel ist. Es kann mit der Erweiterung LUKS (*Linux Unified Key Setup*) verwendet werden, einer Spezifikation, die einen plattformunabhängigen Standard für die Verwendung mit verschiedenen Tools implementiert.

Für die Stacked-Methode sollten Sie einen Blick auf *EncFS* werfen, das wahrscheinlich der einfachste Weg ist, Daten unter Linux zu sichern, da es keine Root-Rechte benötigt und auf einem bestehenden Dateisystem ohne Änderungen funktioniert.

Wenn Sie auf Daten auf verschiedenen Plattformen zugreifen müssen, schauen Sie sich schließlich *Veracrypt* an, den Nachfolger von Truecrypt. Es erstellt verschlüsselte Medien und Dateien, die sowohl unter Linux als auch unter macOS und Windows verwendet werden können.

Geführte Übungen

1. Verwenden Sie in Ihrem Browser ein “privates Fenster”, um:

völlig anonym im Internet zu surfen	
keine Spuren auf dem Computer zu hinterlassen, den Sie verwenden	
TLS zu aktivieren, um das Verfolgen von Cookies zu vermeiden	
DNT zu verwenden	
Kryptographie bei der Datenübertragung zu nutzen	

2. Was ist OpenStack?

Ein Projekt für den Aufbau von privatem IaaS	
Ein Projekt für den Aufbau von privatem PaaS	
Ein Projekt für den Aufbau von privatem SaaS	
Ein Hypervisor	
Ein Open-Source-Passwortmanager	

3. Welche der folgenden Optionen sind gültige Festplattenverschlüsselungssoftware?

RevealJS, EncFS und dm-crypt	
dm-crypt und KeePass	
EncFS und Bitwarden	
EncFS und dm-crypt	
TLS und dm-crypt	

4. Wählen Sie wahr oder falsch für die dm-crypt Geräteverschlüsselung:

Dateien werden verschlüsselt, bevor sie auf die Festplatte geschrieben werden	
Das gesamte Dateisystem ist ein verschlüsselter Blob	

Es werden nur Dateien und Verzeichnisse verschlüsselt, nicht Symlinks	
Kein Root-Zugriff erforderlich	
Ist eine Block-Device-Verschlüsselung	

5. Beamer ist:

Ein Verschlüsselungsmechanismus	
Ein Hypervisor	
Eine Virtualisierungssoftware	
Eine OpenStack-Komponente	
Ein LaTeX-Präsentations-Tool	

Offene Übungen

1. Die meisten Distributionen werden standardmäßig mit Firefox installiert (ist das bei Ihnen nicht der Fall, müssen Sie Firefox zuerst installieren). Wir werden eine Firefox-Erweiterung namens *Lightbeam* installieren, indem Sie entweder `Strg` + `Shift` + `A` drücken und "Lightbeam" in das Suchfeld eingeben, das auf dem geöffneten Tab angezeigt wird, oder indem Sie die Erweiterungsseite mit Firefox aufrufen und auf die Schaltfläche "Install" klicken: https://addons.mozilla.org/en-US/firefox/addon/lightbeam. Anschließend starten Sie die Erweiterung, indem Sie auf das Symbol klicken, und besuchen Sie einige Webseiten auf anderen Tabs, um zu sehen, was passiert.
2. Was ist das Wichtigste bei der Verwendung eines Passwortmanagers?

3. Verwenden Sie Ihren Webbrowser, um https://haveibeenpwned.com/ aufzurufen. Finden Sie den Zweck der Website heraus und überprüfen Sie, ob Ihre E-Mail-Adresse in einigen Datenlecks enthalten war.

Zusammenfassung

Das Terminal ist eine leistungsstarke Möglichkeit, mit dem System zu interagieren, und es gibt viele nützliche und sehr ausgereifte Tools, die Sie in dieser Umgebung nutzen können. Sie gelangen zum Terminal, indem Sie in Ihrer Desktop-Umgebung nach einem grafischen Terminal suchen oder kbd drücken:[Strg+Alt+F#].

Linux wird hauptsächlich in der Technologiebranche eingesetzt, um IaaS-, PaaS- und SaaS-Dienste anzubieten, wobei vor allem drei Hypervisoren eine wichtige Rolle bei der Unterstützung spielen: Xen, KVM und Virtualbox.

Der Browser ist heute ein unverzichtbares Werkzeug auf Computern, aber man muss einige Dinge verstehen, um ihn sicher zu benutzen. DNT ist nur eine Möglichkeit, einer Website zu signalisieren, dass Sie nicht verfolgt werden wollen, aber es gibt keine Garantie. Private Fenster sind nur für den Computer, den Sie benutzen, privat, aber genau das kann Sie vor Cookie Tracking bewahren.

TLS kann Ihre Kommunikation im Internet verschlüsseln, aber Sie müssen erkennen, wann es verwendet wird. Starke Passwörter sind zur Absicherung ebenfalls sehr wichtig, weshalb Sie diese Verantwortung am besten einem Passwortmanager übertragen und der Software erlauben, Zufallspasswörter für jede Website zu erstellen, an der Sie sich anmelden.

Eine weitere Möglichkeit, Ihre Kommunikation zu sichern, besteht darin, Ihre Dateiordner und E-Mails mit GnuPG zu signieren und zu verschlüsseln. dm-crypt und EncFS sind zwei Alternativen zur Verschlüsselung ganzer Festplatten oder Partitionen, die Block- bzw. Stack-Verschlüsselung verwenden.

Schließlich ist LibreOffice Impress eine sehr vollständige Open-Source-Alternative zu Microsoft Powerpoint, aber es gibt Beamer und RevealJS, wenn Sie Präsentationen lieber mit Code statt mit GUIs erstellen. ProjectLibre und GanttProject können die richtige Wahl sein, wenn Sie einen Ersatz für Microsoft Project benötigen.

Lösungen zu den geführten Übungen

1. Verwenden Sie in Ihrem Browser ein “privates Fenster”, um:

völlig anonym im Internet zu surfen	
keine Spuren auf dem Computer zu hinterlassen, den Sie verwenden	X
TLS zu aktivieren, um das Verfolgen von Cookies zu vermeiden	
DNT zu verwenden	
Kryptographie bei der Datenübertragung zu nutzen	

2. Was ist OpenStack?

Ein Projekt für den Aufbau von privatem IaaS	X
Ein Projekt für den Aufbau von privatem PaaS	
Ein Projekt für den Aufbau von privatem SaaS	
Ein Hypervisor	
Ein Open-Source-Passwortmanager	

3. Welche der folgenden Optionen sind gültige Festplattenverschlüsselungssoftware?

RevealJS, EncFS und dm-crypt	
dm-crypt und KeePass	
EncFS und Bitwarden	
EncFS und dm-crypt	X
TLS und dm-crypt	

4. Wählen Sie wahr oder falsch für die dm-crypt Geräteverschlüsselung:

Dateien werden verschlüsselt, bevor sie auf die Festplatte geschrieben werden	wahr
Das gesamte Dateisystem ist ein verschlüsselter Blob	wahr

Es werden nur Dateien und Verzeichnisse verschlüsselt, nicht Symlinks	falsch
Kein Root-Zugriff erforderlich	falsch
Ist eine Block-Device-Verschlüsselung	wahr

5. Beamer ist:

Ein Verschlüsselungsmechanismus	
Ein Hypervisor	
Eine Virtualisierungssoftware	
Eine OpenStack-Komponente	
Ein LaTeX-Präsentations-Tool	X

Lösungen zu den offenen Übungen

1. Die meisten Distributionen werden standardmäßig mit Firefox installiert (ist das bei Ihnen nicht der Fall, müssen Sie Firefox zuerst installieren). Wir werden eine Firefox-Erweiterung namens *Lightbeam* installieren, indem Sie entweder `Strg` + `Shift` + `A` drücken und "Lightbeam" in das Suchfeld eingeben, das auf dem geöffneten Tab angezeigt wird, oder indem Sie die Erweiterungsseite mit Firefox aufrufen und auf die Schaltfläche "Install" klicken: https://addons.mozilla.org/en-US/firefox/addon/lightbeam. Anschließend starten Sie die Erweiterung, indem Sie auf das Symbol klicken, und besuchen Sie einige Webseiten auf anderen Tabs, um zu sehen, was passiert.

   Erinnern Sie sich an die erwähnten Cookies, die Ihre Daten mit verschiedenen Diensten teilen können, wenn Sie eine Website besuchen? Das ist genau das, was Ihnen diese Erweiterung anzeigt. Lightbeam ist ein Mozilla-Experiment, das versucht, diese und die Websites Dritter offenzulegen, mit denen Sie beim Besuch einer einzelnen URL interagieren. Das ist normalerweise für den Benutzer nicht sichtbar und kann zeigen, dass eine einzelne Website manchmal mit einem Dutzend oder mehr Diensten interagiert.

2. Was ist das Wichtigste bei der Verwendung eines Passwortmanagers?

   Beim Einsatz eines Passwortmanagers ist es am wichtigsten, sich das Master-Passwort zu merken und ein einmaliges Zufallspasswort für jeden einzelnen Dienst zu verwenden.

3. Verwenden Sie Ihren Webbrowser, um https://haveibeenpwned.com/ aufzurufen. Finden Sie den Zweck der Website heraus und überprüfen Sie, ob Ihre E-Mail-Adresse in einigen Datenlecks enthalten war.

   Die Website unterhält eine Datenbank mit Login-Informationen, deren Passwörter von einem Passwortleck betroffen waren. Sie bietet die Suche nach einer E-Mail-Adresse und zeigt an, ob diese E-Mail-Adresse in einer öffentlichen Datenbank mit gestohlenen Anmeldeinformationen enthalten war. Es besteht die Möglichkeit, dass auch Ihre E-Mail-Adresse von dem einen oder anderen Leck betroffen ist. Wenn dies der Fall ist, stellen Sie sicher, dass Sie Ihre Passwörter kürzlich aktualisiert haben. Wenn Sie noch keinen Passwortmanager verwenden, werfen Sie einen Blick auf die in dieser Lektion empfohlenen.

Linux Professional Institute

Thema 2: Sich auf einem Linux-System zurechtfinden

2.1 Grundlagen der Befehlszeile

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 2.1

Gewichtung

3

Hauptwissensgebiete

- Shell-Grundlagen
- Befehlszeilen-Syntax
- Variablen
- Quoting

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- Bash
- `echo`
- `history`
- Umgebungsvariable `PATH`
- `export`
- `type`

2.1 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.1 Grundlagen der Befehlszeile
Lektion:	1 von 2

Einführung

Moderne Linux-Distributionen stellen eine Vielzahl grafischer Benutzeroberflächen bereit, aber ein Administrator muss stets wissen, wie man mit der Befehlszeile, auch *Shell* genannt, arbeitet. Die Shell ist ein Programm, das eine textbasierte Kommunikation zwischen dem Betriebssystem und dem Benutzer ermöglicht.

Es gibt verschiedene Shells unter Linux, hier nur einige wenige:

- Bourne-again shell (Bash)
- C shell (csh oder tcsh, eine erweiterte csh)
- Korn shell (ksh)
- Z shell (zsh)

Die meistgenutzte Shell unter Linux ist die Bash, die auch in den folgenden Beispielen und Übungen zum Einsatz kommt.

Bei der Arbeit mit einer interaktiven Shell gibt der Benutzer Befehle an der sogenannten *Eingabeaufforderung* oder *Prompt* ein. In jeder Linux-Distribution kann der Standard-Prompt

etwas anders aussehen, folgt aber in der Regel dieser Struktur:

```
username@hostname aktuelles_verzeichnis shell_typ
```

Unter Ubuntu oder Debian GNU/Linux wird die Eingabeaufforderung für einen normalen Benutzer wahrscheinlich so aussehen:

```
carol@mycomputer:~$
```

Der Prompt des Superusers sieht dann so aus:

```
root@mycomputer:~#
```

Unter CentOS oder Red Hat Linux sieht die Eingabeaufforderung für einen normalen Benutzer hingegen so aus:

```
[dave@mycomputer ~]$
```

Und der Prompt des Superusers sieht so aus:

```
[root@mycomputer ~]#
```

Schauen wir uns die einzelnen Komponenten dieser Struktur an:

`username`

Name des Benutzers, der die Shell ausführt

`hostname`

Name des Hosts, auf dem die Shell läuft. Es gibt auch den Befehl `hostname`, mit dem man den Hostname eines Systems anzeigt oder ändert.

`aktuelles_verzeichnis`

Das Verzeichnis, in dem sich die Shell gerade befindet. Die Tilde (`~`) bedeutet, dass sich die Shell im Heimatverzeichnis des aktuellen Benutzers befindet.

`shell_typ`

`$` zeigt an, dass die Shell von einem normalen Benutzer ausgeführt wird.

`#` zeigt an, dass die Shell vom Superuser `root` ausgeführt wird.

Da wir keine besonderen Privilegien benötigen, werden wir in den folgenden Beispielen einen unprivilegierten Prompt verwenden; der besseren Übersicht halber verwenden wir einfach `$` als Prompt.

Aufbau der Befehlszeile

Die meisten Befehle auf der Befehlszeile folgen derselben Grundstruktur:

```
Befehl  [Option(en)/Parameter...]  [Argument(e)...]
```

Betrachten wir examplarisch den folgenden Befehl:

```
$ ls -l /home
```

Zu den einzelnen Bestandteilen:

Befehl

Programm, das der Benutzer ausführt — im obigen Beispiel `ls`.

Option(en)/Parameter

Ein "`Schalter`", der das Verhalten des Befehls in irgendeiner Weise verändert, z.B. `-l` im obigen Beispiel. Auf Optionen kann in Kurz- oder Langform zugegriffen werden, z.B. ist `-l` identisch mit `--format=long`.

Mehrere Optionen lassen sich kombinieren, und für die Kurzform können die Buchstaben in der Regel zusammengeschrieben werden. Beispielsweise machen die folgenden Befehle alle dasselbe:

```
$ ls -al
$ ls -a -l
$ ls --all --format=long
```

Argument(e)

Zusätzliche Angaben, die vom Programm benötigt werden, etwa Dateiname oder Pfad (`/home` im obigen Beispiel).

Der einzige obligatorische Teil dieser Struktur ist der Befehl selbst. Im Allgemeinen sind alle

anderen Elemente optional, aber ein Programm kann die Angabe bestimmter Optionen, Parameter oder Argumente erfordern.

NOTE Die meisten Befehle liefern einen Überblick über mögliche Optionen, wenn sie mit dem Parameter `--help` aufgerufen werden. Weitere Informationsquellen zu einzelnen Linux-Befehlen werden wir noch kennenlernen.

Befehlstypen

Die Shell unterstützt zwei Arten von Befehlen:

Interne Befehle (Builtins)

Diese sind Teil der Shell selbst und keine eigenständigen Programme. Es gibt etwa 30 solcher Befehle, deren Hauptzweck es ist, Aufgaben innerhalb der Shell auszuführen (z.B. `cd`, `set`, `export`).

Externe Befehle

Diese befinden sich in einzelnen Dateien. In der Regel sind es binäre Programme oder Skripte. Wird ein Befehl ausgeführt, der kein Builtin ist, sucht die Shell mit der Variablen `PATH` nach einer ausführbaren Datei mit dem Namen des Befehls.

Der Befehl `type` zeigt, welchen Typs ein bestimmter Befehl ist:

```
$ type echo
echo is a shell builtin
$ type man
man is /usr/bin/man
```

Quoting

Als Linux-Nutzer müssen Sie Dateien oder Variablen auf verschiedene Weise erstellen oder manipulieren, was bei der Arbeit mit kurzen Dateinamen und einzelnen Werten einfach ist. Es wird allerdings komplizierter, wenn Leerzeichen, Sonderzeichen und Variablen im Spiel sind. Shells bieten eine Funktion namens *Quoting*, die solche Daten mit verschiedenen Arten von Anführungszeichen kapselt (" ", ' '). In Bash gibt es drei Arten von Anführungszeichen:

- Doppelte Anführungszeichen
- Einfache Anführungszeichen
- Escape-Zeichen

Beispielsweise verhalten sich die folgenden Befehle aufgrund von Quoting nicht in der gleichen Weise:

```
$ TWOWORDS="two words"
$ touch $TWOWORDS
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 two
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 words
$ touch "$TWOWORDS"
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
$ touch '$TWOWORDS'
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 15:00 '$TWOWORDS'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
```

NOTE Die Zeile mit `TWOWORDS=` ist eine Bash-Variable, die wir selbst erstellt haben. Wir werden die Variablen später einführen. Dies soll Ihnen nur zeigen, wie sich das Quoting auf die Ausgabe von Variablen auswirkt.

Doppelte Anführungszeichen

Doppelte Anführungszeichen weisen die Shell an, den Text zwischen den Anführungszeichen ("....") als reguläre Zeichen zu übernehmen; alle Sonderzeichen verlieren ihre Bedeutung — mit Ausnahme von `$` (Dollarzeichen), `\` (Backslash) und `` ` `` (Backquote), so dass Variablen, Befehlsersetzung und arithmetische Funktionen weiterhin verwendet werden können.

So wird beispielsweise die Ersetzung der Variablen `$USER` durch die doppelten Anführungszeichen nicht beeinflusst:

```
$ echo I am $USER
I am tom
$ echo "I am $USER"
I am tom
```

Ein Leerzeichen hingegen verliert seine Bedeutung als Argumententrenner:

```
$ touch new file
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 file
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 new
$ touch "new file"
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:19 new file
```

Wie Sie sehen, erzeugt der Befehl `touch` im ersten Beispiel zwei einzelne Dateien; der Befehl interpretiert die beiden Zeichenketten als einzelne Argumente. Im zweiten Beispiel interpretiert der Befehl beide Zeichenketten als *ein* Argument, also nur eine Datei. Sie sollten allerdings Leerzeichen in Dateinamen vermeiden und stattdessen einen Unterstrich (`_`) oder einen Punkt (`.`) verwenden.

Einfache Anführungszeichen

Einfache Anführungszeichen kennen keine Ausnahmen wie die doppelten Anführungszeichen: Sie widerrufen jede spezielle Bedeutung für jedes Zeichen. Nehmen wir eines der Beispiele von oben:

```
$ echo I am $USER
I am tom
```

Bei der Verwendung der einfachen Anführungszeichen sehen Sie ein anderes Ergebnis:

```
$ echo 'I am $USER'
I am $USER
```

Der Befehl zeigt nun die Zeichenkette exakt an, ohne die Variable zu ersetzen.

Escape-Zeichen

Wir können *Escape-Zeichen* (Escape Characters) nutzen, um spezielle Bedeutungen von Zeichen aus der Bash zu entfernen. Zurück zur Umgebungsvariablen `$USER`:

```
$ echo $USER
carol
```

Wir sehen, dass standardmäßig der Inhalt der Variable im Terminal angezeigt wird. Wenn wir

jedoch dem Dollar-Zeichen ein Backslash-Zeichen (`\`) voranstellen, wird die besondere Bedeutung des Dollar-Zeichens aufgehoben. Dies wiederum lässt Bash den Wert der Variablen nicht auf den Benutzernamen der Person, die den Befehl ausführt, erweitern, sondern interpretiert den Variablennamen wörtlich:

```
$ echo \$USER
$USER
```

Wenn Sie sich erinnern, können wir mit einfachen Anführungszeichen ein ähnliches Ergebnis erzielen, da sie dafür sorgen, dass der Inhalt zwischen den Anführungszeichen zeichengenau ausgegeben wird. Das Escape-Zeichen funktioniert jedoch anders, indem es Bash anweist, jede spezielle Bedeutung, die das nachfolgende Zeichen haben könnte, zu ignorieren.

Geführte Übungen

1. Teilen Sie die folgenden Zeilen in die Bestandteile Befehl, Option(en)/Parameter und Argument(e) auf:
   - Beispiel: `cat -n /etc/passwd`

Befehl:	`cat`
Option:	`-n`
Argument:	`/etc/passwd`

   - `ls -l /etc`

Befehl:	
Option:	
Argument:	

   - `ls -l -a`

Befehl:	
Option:	
Argument:	

   - `cd /home/user`

Befehl:	
Option:	
Argument:	

2. Bestimmen Sie den Befehlstyp:

   Beispiel:

`pwd`	Shell-Builtin
`mv`	Externer Befehl

`cd`	

`cat`	
`exit`	

3. Lösen Sie die folgenden Befehle mit Anführungszeichen auf:

   Beispiel:

`echo "$HOME is my home directory"`	`echo /home/user is my home directory`

`touch "$USER"`	
`touch 'touch'`	

Offene Übungen

1. Legen Sie mit einem Befehl und unter Verwendung der Klammer-Erweiterung (*Brace Expansion*) in der Bash (siehe Manpage der Bash) 5 von 1 bis 5 nummerierte Dateien mit dem Präfix `game` an (`game1`, `game2`,...).

2. Löschen Sie alle 5 Dateien, die Sie gerade mit nur einem Befehl erstellt haben, unter Verwendung eines anderen Sonderzeichens (siehe *Pathname Expansion* in den Man Pages der Bash).

3. Gibt es andere Möglichkeiten, zwei Befehle miteinander interagieren zu lassen? Welche sind das?

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Konzepte der Linux-Shell
- Was ist die Bash-Shell
- Die Struktur der Kommandozeile
- Eine Einführung ins Quoting

Befehle, die in den Übungen verwendet werden:

`bash`
Die beliebteste Shell auf Linux-Rechnern.

`echo`
Gibt Text im Terminal aus.

`ls`
Listet den Inhalt eines Verzeichnisses auf.

`type`
Zeigt an, wie ein bestimmter Befehl ausgeführt wird.

`touch`
Erstellt eine leere Datei oder aktualisiert das Änderungsdatum einer bestehenden Datei.

`hostname`
Zeigt oder ändert den Hostnamen eines Systems.

Lösungen zu den geführten Übungen

1. Teilen Sie die folgenden Zeilen in die Bestandteile Befehl, Option(en)/Parameter und Argument(e) auf:
   - `ls -l /etc`

Befehl:	`ls`
Option:	`-l`
Argument:	`/etc`

   - `ls -l -a`

Befehl:	`ls`
Option:	`-l -a`
Argument:	

   - `cd /home/user`

Befehl:	`cd`
Option:	
Argument:	`/home/user`

2. Bestimmen Sie den Befehlstyp:

`cd`	Shell builtin
`cat`	External command
`exit`	Shell builtin

3. Lösen Sie die folgenden Befehle mit Anführungszeichen auf:

`touch "$USER"`	`tom`
`touch 'touch'`	Creates a file named `touch`

Lösungen zu den offenen Übungen

1. Legen Sie mit einem Befehl und unter Verwendung der Klammer-Erweiterung (*Brace Expansion*) in der Bash (siehe Manpage der Bash) 5 von 1 bis 5 nummerierte Dateien mit dem Präfix `game` an (`game1`, `game2`,...).

   Bereiche können verwendet werden, um die Zahlen von 1 bis 5 innerhalb eines Befehls auszudrücken:

   ```
   $ touch game{1..5}
   $ ls
   game1  game2  game3  game4  game5
   ```

2. Löschen Sie alle 5 Dateien, die Sie gerade mit nur einem Befehl erstellt haben, unter Verwendung eines anderen Sonderzeichens (siehe *Pathname Expansion* in den Man Pages der Bash).

   Da alle Dateien mit `game` beginnen und mit einem einzigen Zeichen enden (in diesem Fall einer Zahl von 1 bis 5), kann `?` als Sonderzeichen für das letzte Zeichen im Dateinamen verwendet werden:

   ```
   $ rm game?
   ```

3. Gibt es andere Möglichkeiten, zwei Befehle miteinander interagieren zu lassen? Welche sind das?

   Ja, ein Befehl könnte z.B. Daten in eine Datei schreiben, die dann von einem anderen Befehl verarbeitet wird. Linux kann auch die Ausgabe eines Befehls sammeln und als Eingabe für einen anderen Befehl verwenden; das heißt *Piping* und wird in einer späteren Lektion behandelt.

2.1 Lektion 2

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.1 Grundlagen der Befehlszeile
Lektion:	2 von 2

Einführung

Alle Shells verwalten während der Shell-Sitzungen einen Satz von Statusinformationen, die sich während der Sitzung ändern können und das Verhalten der Shell beeinflussen. Programme nutzen diese Daten, um Aspekte der Systemkonfiguration zu bestimmen. Die meisten dieser Daten werden in sogenannten *Variablen* gespeichert, die wir in dieser Lektion behandeln.

Variablen

Variablen sind Speicher für Daten, beispielsweise Text oder Zahlen, auf die später zugegriffen werden kann. Variablen haben einen Namen, über den man auf sie zugreift, auch wenn sich ihr Inhalt ändert. Sie sind zudem ein wichtiges Werkzeug in den meisten Programmiersprachen.

In den meisten Linux-Shells gibt es zwei Arten von Variablen:

Lokale Variablen

Diese Variablen stehen nur in der jeweiligen Shell-Sitzung zur Verfügung. Wenn Sie eine lokale Variable erstellen und dann ein anderes Programm von dieser Shell aus starten, ist die Variable nicht mehr zugänglich. Da sie nicht an Subprozesse vererbt werden, nennt man diese

Variablen *lokale Variablen* (*local variables*).

Umgebungsvariablen

Diese Variablen stehen sowohl in einer Shell-Sitzung als auch in Unterprozessen zur Verfügung, die aus dieser Shell-Sitzung hervorgegangen sind. Diese Variablen werden etwa genutzt, um Konfigurationsdaten an Befehle zu übergeben. Da Programme auf diese Variablen zugreifen können, heißen sie *Umgebungsvariablen* (*environment variables*). Die Mehrheit der Umgebungsvariablen ist in Großbuchstaben geschrieben (beispielsweise `PATH`, `DATE`, `USER`). Ein Satz von Standardumgebungsvariablen liefert zum Beispiel Informationen über das Home-Verzeichnis oder den Terminaltyp des Benutzers.

NOTE Variablen sind nicht persistent. Wird die Shell, in der sie gesetzt wurden, geschlossen, gehen alle Variablen und deren Inhalt verloren. Die meisten Shells stellen Konfigurationsdateien mit Variablen bereit, die beim Start einer neuen Shell gesetzt werden. Variablen, die dauerhaft gesetzt werden sollen, müssen zu einer dieser Konfigurationsdateien hinzugefügt werden.

Manipulation von Variablen

Als Systemadministrator müssen Sie sowohl lokale als auch Umgebungsvariablen erzeugen, ändern oder entfernen.

Arbeiten mit lokalen Variablen

Sie setzen eine lokale Variable mit dem Operator `=` (Gleichheitszeichen). Eine einfache Zuweisung erstellt eine lokale Variable:

```
$ greeting=hello
```

NOTE Setzen Sie keine Leerstelle vor oder nach dem `=` Operator!

Sie können jede Variable mit dem Befehl `echo` anzeigen, der normalerweise den Text im Argumentabschnitt anzeigt:

```
$ echo greeting
greeting
```

Um auf den Wert der Variablen zuzugreifen, müssen Sie ein `$` (Dollarzeichen) vor den Variablennamen setzen.

```
$ echo $greeting
hello
```

Wie man sieht, wurde die Variable erzeugt. Öffnen Sie nun eine weitere Shell und versuchen Sie, den Inhalt der Variable anzuzeigen:

```
$ echo $greeting
```

Es wird nichts angezeigt. Hier wird deutlich, dass Variablen immer nur in einer bestimmten Shell existieren.

Um zu überprüfen, ob es sich bei der Variablen tatsächlich um eine lokale Variable handelt, starten Sie einen neuen Prozess und prüfen Sie, ob dieser Prozess auf die Variable zugreifen kann. Starten Sie dazu eine andere Shell und führen Sie darin den Befehl `echo` aus. Da die neue Shell in einem neuen Prozess läuft, erbt sie keine lokalen Variablen von ihrem übergeordneten Prozess:

```
$ echo $greeting world
hello world
$ bash -c 'echo $greeting world'
world
```

NOTE Nutzen Sie einfache Anführungszeichen in dem Beispiel oben.

Um eine Variable zu entfernen, nutzen Sie den Befehl `unset`:

```
$ echo $greeting
hey
$ unset greeting
$ echo $greeting
```

NOTE `unset` verlangt den Namen der Variablen als Argument. Daher dürfen Sie dem Namen kein `$` voranstellen, da dies die Variable auflösen und den Wert der Variablen an `unset` anstelle des Namens der Variablen übergeben würde.

Arbeiten mit Umgebungsvariablen

Um eine Variable für Unterprozesse verfügbar zu machen, verwandeln Sie sie von einer lokalen in eine globale oder Umgebungsvariable, und zwar mit dem Befehl `export`. Wird sie über den Variablennamen aufgerufen, wird diese Variable der Umgebung der Shell hinzugefügt:

```
$ greeting=hello
$ export greeting
```

NOTE Noch einmal: Stellen Sie sicher, dass Sie kein `$` beim Aufruf von `export` verwenden, da Sie den Variablennamen, nicht den Inhalt der Variablen übergeben wollen.

Eine einfachere Möglichkeit, die Umgebungsvariable zu erstellen, besteht darin, beide der oben genannten Methoden zu kombinieren, indem man den Variablenwert im Argumentteil des Befehls zuweist.

```
$ export greeting=hey
```

Lassen Sie uns noch einmal überprüfen, ob die Variable für Unterprozesse zugänglich ist:

```
$ export greeting=hey
$ echo $greeting world
hey world
$ bash -c 'echo $greeting world'
hey world
```

Eine weitere Einsatzmöglichkeit von Umgebungsvariablen besteht darin, sie vor Befehle zu setzen. Testen wir das mit der Umgebungsvariablen `TZ`, die die Zeitzone enthält. Wir nutzen die Variable, um dem Befehl `date` anzuzeigen, welche Zeitzone er nutzen soll:

```
$ TZ=EST date
Thu 31 Jan 10:07:35 EST 2019
$ TZ=GMT date
Thu 31 Jan 15:07:35 GMT 2019
```

Mit dem Befehl `env` zeigen Sie alle Umgebungsvariablen an.

Die Variable `PATH`

Die Variable `PATH` ist eine der wichtigsten Umgebungsvariablen in einem Linux-System. Sie speichert eine Liste von durch Doppelpunkt getrennten Verzeichnissen mit ausführbaren Programmen, die als Befehle aus der Linux-Shell aufrufbar sind.

```
$ echo $PATH
```

```
/home/user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Um ein neues Verzeichnis an die Variable anzuhängen, setzen Sie einen Doppelpunkt (`:`):

```
$ PATH=$PATH:new_directory
```

Hier ein Beispiel:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ PATH=$PATH:/home/user/bin
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin
```

Wie Sie sehen, wird `$PATH` mit dem neuen Wert verwendet, der `PATH` zugewiesen wurde. Die Variable wird während der Befehlsausführung aufgelöst und sorgt dafür, dass der ursprüngliche Inhalt der Variablen erhalten bleibt. Sie können diese Zuweisung natürlich auch bei anderen Variablen durchführen.

```
$ mybin=/opt/bin
$ PATH=$PATH:$mybin
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin:/opt/bin
```

Die Variable `PATH` ist mit Vorsicht zu behandeln, da sie für die Arbeit auf der Kommandozeile entscheidend ist. Betrachten wir die folgende Variable `PATH`:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Um herauszufinden, wie die Shell einen bestimmten Befehl aufruft, führen wir `which` mit dem Namen des Befehls als Argument aus. So ermitteln wir z.B., wo `nano` gespeichert ist:

```
$ which nano
/usr/bin/nano
```

Offenkundig befindet sich die ausführbare Datei `nano` im Verzeichnis `/usr/bin`. Entfernen wir das Verzeichnis aus der Variablen und überprüfen wir, ob der Befehl noch funktioniert:

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
```

Schauen wir noch einmal nach dem Befehl `nano`:

```
$ which nano
which: no nano in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games)
```

Offensichtlich wird der Befehl nicht gefunden, also nicht ausgeführt. Die Fehlermeldung nennt auch den Grund, warum der Befehl nicht gefunden und an welchen Stellen er gesucht wurde.

Fügen wir die Verzeichnisse wieder hinzu und versuchen, den Befehl erneut auszuführen.

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ which nano
/usr/bin/nano
```

Jetzt funktioniert unser Befehl wieder.

TIP Die Reihenfolge der Elemente in `PATH` definiert auch die Reihenfolge der Suche: Die erste passende ausführbare Datei, die beim Durchlaufen der Pfade gefunden wird, wird ausgeführt.

Geführte Übungen

1. Erzeugen Sie die lokale Variable `number`.

2. Erzeugen Sie die Umgebungsvariable `ORDER` mit Hilfe der beiden oben genannten Methoden.

3. Lassen Sie sowohl Namen als auch Inhalt der Variablen anzeigen.

4. Welche Reichweiten (Scope) haben die zuvor erzeugten Variablen?

Offene Übungen

1. Erzeugen Sie eine lokale Variable `nr_files` und weisen Sie die Anzahl der Zeilen in der Datei `/etc/passwd` zu. Hinweis: Schauen Sie sich den Befehl `wc` und die Befehlsersetzung an und vergessen Sie nicht die Anführungszeichen.

2. Erzeugen Sie eine Umgebungsvariable `ME`. Weisen Sie `USER` als Wert zu.

3. Fügen Sie den Wert der Variablen `HOME` an `ME` mit dem Trennzeichen `:` an und zeigen Sie den Inhalt der Variablen `ME` an.

4. Erstellen Sie unter Verwendung des obigen Datumsbeispiels eine Variable namens `today` und weisen Sie das Datum für eine Zeitzone zu.

5. Erzeugen Sie eine weitere Variable namens `today1` und weisen Sie ihr das Systemdatum zu.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Arten von Variablen
- Wie man Variablen erzeugt
- Wie man Variablen manipuliert

Befehle, die in den Übungen verwendet werden:

`env`
: Zeigt die aktuelle Umgebung an.

`echo`
: Gibt Text aus.

`export`
: Macht lokale Variablen für Unterprozesse verfügbar.

`unset`
: Entfernt eine Variable.

Lösungen zu den geführten Übungen

1. Erzeugen Sie die lokale Variable `number`.

```
$ number=5
```

2. Erzeugen Sie die Umgebungsvariable `ORDER` mit Hilfe der beiden oben genannten Methoden.

```
$ export ORDER=desc
```

3. Lassen Sie sowohl Namen als auch Inhalt der Variablen anzeigen.

```
$ echo number
number
$ echo ORDER
ORDER
$ echo $number
5
$ echo $ORDER
desc
```

4. Welche Reichweiten (Scope) haben die zuvor erzeugten Variablen?
   - Der Scope der lokalen Variablen `number` ist nur die aktuelle Shell.
   - Der Scope der Umgebungsvariablen `ORDER` sind die aktuelle Shell und alle von dieser erzeugten Subshells.

Lösungen zu den offenen Übungen

1. Erzeugen Sie eine lokale Variable `nr_files` und weisen Sie die Anzahl der Zeilen in der Datei `/etc/passwd` zu. Hinweis: Schauen Sie sich den Befehl `wc` und die Befehlsersetzung an und vergessen Sie nicht die Anführungszeichen.

```
$ nr_files=`wc -l /etc/passwd`
```

2. Erzeugen Sie eine Umgebungsvariable `ME`. Weisen Sie `USER` als Wert zu.

```
$ export ME=$USER
```

3. Fügen Sie den Wert der Variablen `HOME` an `ME` mit dem Trennzeichen `:` an und zeigen Sie den Inhalt der Variablen `ME` an.

```
$ ME=$ME:$HOME
$ echo $ME
user:/home/user
```

4. Erstellen Sie unter Verwendung des obigen Datumsbeispiels eine Variable namens `today` und weisen Sie das Datum für eine Zeitzone zu.

   Im Folgenden werden die Zeitzonen GMT und EST als Beispiele verwendet, aber jede Zeitzonenauswahl ist gültig.

```
$ today=$(TZ=GMT date)
$ echo $today
Thu 31 Jan 15:07:35 GMT 2019
```

   or

```
$ today=$(TZ=EST date)
$ echo $today
Thu 31 Jan 10:07:35 EST 2019
```

5. Erzeugen Sie eine weitere Variable namens `today1` und weisen Sie ihr das Systemdatum zu.

   Nehmen wir an, Sie befinden sich in GMT:

```
$ today1=$(date)
$ echo $today1
Thu 31 Jan 10:07:35 EST 2019
```

2.2 Hilfe suchen über die Befehlszeile

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 2.2

Gewichtung

2

Hauptwissensgebiete

- Manpages
- Infopages

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

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

2.2 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.2 Hilfe suchen über die Befehlszeile
Lektion:	1 von 1

Einführung

Die Kommandozeile ist ein sehr komplexes Werkzeug. Jeder Befehl hat seine eigenen Optionen, und darum ist Dokumentation der Schlüssel zu einem Linux-System. Neben dem Verzeichnis `/usr/share/doc/`, in dem die meiste Dokumentation liegt, bieten verschiedene andere Tools Informationen zur Verwendung von Linux-Befehlen. Diese Lektion konzentriert sich auf Methoden zum Zugriff auf diese Dokumentation, um Hilfe zu erhalten.

Es gibt eine Vielzahl von Methoden, um Hilfe auf der Linux-Befehlszeile zu erhalten: `man`, `help` und `info` sind nur einige davon. Für Linux Essentials konzentrieren wir uns auf `man` und `info`, da es sich um die am häufigsten verwendeten Hilfe-Tools handelt.

Ein weiteres Thema dieser Lektion ist das Auffinden von Dateien, wobei Sie hauptsächlich mit dem Befehl `locate` arbeiten werden.

Hilfe auf der Kommandozeile aufrufen

Eingebaute Hilfe

Mit dem Parameter `--help` aufgerufen, liefern die meisten Befehle eine kurze Übersicht zu ihrer Nutzung. Obwohl nicht alle Befehle diesen Schalter bereitstellen, ist es dennoch ein guter erster Versuch, mehr über die Parameter eines Befehls zu erfahren. Beachten Sie , dass die Anweisungen von `--help` im Vergleich zu anderen Dokumentationsquellen, die wir im weiteren Verlauf dieser Lektion besprechen werden, oft recht knapp gehalten sind.

Manpages

Die meisten Befehle bieten eine “Manual Page” oder kurz “Manpage”. Diese Dokumentation wird in der Regel mit der Software installiert und kann mit dem Befehl `man` aufgerufen werden: Der Befehl, dessen Manpage angezeigt werden soll, wird `man` als Argument mitgegeben:

```
$ man mkdir
```

Dieser Befehl öffnet die Manpage für `mkdir`. Über die Pfeiltasten nach oben und unten oder die Leertaste navigieren Sie durch die Manpage — `Q` schließt sie wieder.

Jede Manpage ist in maximal 11 Abschnitte unterteilt, wobei viele dieser Abschnitte optional sind:

Abschnitt	**Beschreibung**
NAME	Name des Befehls und kurze Beschreibung
SYNOPSIS	Beschreibung der Befehlssyntax
DESCRIPTION	Beschreibung der Wirkung des Befehls
OPTIONS	Verfügbare Optionen
ARGUMENTS	Verfügbare Argumente
FILES	Hilfsdateien
EXAMPLES	Ein Beispiel für den Einsatz des Befehls
SEE ALSO	Querverweise zu verwandten Themen
DIAGNOSTICS	Warn- und Fehlermeldungen
COPYRIGHT	Autor(en) des Befehls
BUGS	Bekannte Fehler und Beschränkungen des Befehls

In der Praxis enthalten die meisten Manpages nicht alle diese Teile.

Manpages sind in acht Kategorien organisiert, die von 1 bis 8 nummeriert sind:

Kategorie	**Beschreibung**
1	Benutzerbefehle
2	Systemaufrufe
3	Funktionen der C-Bibliothek
4	Treiber und Gerätedateien
5	Konfigurationsdateien und Dateiformate
6	Spiele
7	Verschiedenes
8	Systemadministrator-Befehle
9	Kernel-Funktionen (nicht Standard)

Jede Manpage gehört zu genau einer Kategorie, mehrere Kategorien können jedoch Manpages mit gleichem Namen enthalten. Nehmen wir beispielweise den Befehl `passwd`, mit dem man das Passwort eines Benutzers ändert: Da `passwd` ein Benutzerbefehl ist, befindet sich seine Manpage in Kategorie 1. Neben dem Befehl `passwd` hat auch die Passwortdatenbankdatei `/etc/passwd` eine Manpage, die ebenfalls `passwd` heißt. Da es sich um eine Konfigurationsdatei handelt, gehört sie zu Kategorie 5. Der Verweis auf eine Manpage enthält darum meist neben dem Befehlsnamen auch den Hinweis auf die entsprechende Kategorie, also beispielsweise `passwd(1)` oder `passwd(5)`.

Standardmäßig zeigt `man passwd` die erste verfügbare Manpage an, in diesem Fall `passwd(1)`. Die Kategorie der gewünschten Manpage ruft man etwa mit `man 1 passwd` oder `man 5 passwd` auf.

Wir haben bereits besprochen, wie man durch eine Manpage navigiert und wie man zur Kommandozeile zurückkehrt. `man` verwendet intern den Befehl `less`, um den Inhalt der Manpage anzuzeigen. `less` lässt Sie nach Text innerhalb einer Manpage suchen. Um nach dem Wort `linux` zu suchen, können Sie einfach `/linux` für die Vorwärtssuche ab dem Punkt, an dem Sie sich auf der Seite befinden, oder `?linux` verwenden, um eine Rückwärtssuche zu starten. Danach sind alle Treffer im Text markiert und die Seite springt zum ersten Treffer. In beiden Fällen springen Sie mit `N` zum nächsten Treffer. Für Informationen zu weiteren Features drücken Sie `H`, und es erscheint ein umfangreiches Menü.

Info-Seiten

Ein weiteres Werkzeug, das Ihnen bei der Arbeit mit dem Linux-System helfen wird, sind die Info-Seiten, die in der Regel detaillierter sind als die Manpages und in Hypertext formatiert sind, ähnlich wie Webseiten im Internet.

Info-Seiten rufen Sie wie folgt auf:

```
$ info mkdir
```

Für jede Info-Seite liest `info` eine Info-Datei, die in einzelne Knoten innerhalb eines Baumes strukturiert ist. Jeder Knoten umfasst ein einfaches Thema, und der Befehl `info` enthält Hyperlinks, über die Sie von einem zum anderen gelangen.

Ähnlich wie `man` hat auch `info` Befehle zur Seitennavigation, über die Sie mehr erfahren, indem Sie auf einer Info-Seite `?` drücken. Diese Werkzeuge erleichtern Ihnen die Navigation und erklären, wie Sie auf die Knoten zugreifen und sich innerhalb des Knotenbaums bewegen.

Das Verzeichnis `/usr/share/doc/`

Wie bereits erwähnt, enthält das Verzeichnis `/usr/share/doc/` die umfangreichste Dokumentation der Befehle, die das System verwendet, sowie je ein Verzeichnis für die meisten der auf dem System installierten Pakete. Der Name eines solchen Verzeichnisses entspricht in der Regel dem Paketnamen, oft ergänzt um die Versionsnummer. Darin findet sich eine Datei `README` oder `readme.txt` mit der grundlegenden Dokumentation des Pakets. Neben der Datei `README` können weitere Dokumentationsdateien enthalten sein, wie etwa das Änderungsprotokoll (Changelog), das die Geschichte des Programms im Detail dokumentiert, oder Beispiele für Konfigurationsdateien.

Die Informationen in der Datei `README` sind von Paket zu Paket unterschiedlich. All diese Dateien sind im Klartext geschrieben, so dass sie mit jedem beliebigen Texteditor zu lesen sind. Anzahl und Art der Dateien hängen vom Paket ab. Schauen Sie sich einige dieser Verzeichnisse an, um einen Überblick über deren Inhalt zu gewinnen.

Dateien suchen

Der Befehl `locate`

Ein Linux-System besteht aus zahlreichen Verzeichnissen und Dateien und verfügt über viele Werkzeuge, um eine bestimmte Datei im System zu finden. Das schnellste ist der Befehl `locate`.

`locate` durchsucht eine Datenbank und gibt dann jeden Namen aus, der mit der angegebenen Zeichenkette übereinstimmt:

```
$ locate note
/lib/udev/keymaps/zepto-znote
/usr/bin/zipnote
/usr/share/doc/initramfs-tools/maintainer-notes.html
/usr/share/man/man1/zipnote.1.gz
```

Der Befehl `locate` unterstützt auch die Verwendung von Wildcards und regulären Ausdrücken, so dass die Suchzeichenfolge nicht mit dem vollständigen Namen der gewünschten Datei übereinstimmen muss. Mehr über reguläre Ausdrücke erfahren Sie in einer späteren Lektion.

Standardmäßig verhält sich `locate` so, als ob das Muster von Asteriks (*) umgeben wäre, so dass `locate MUSTER` das gleiche ist wie `locate *MUSTER*`. Das macht es möglich, Teilzeichenketten anstelle des genauen Dateinamens anzugeben. Sie können das Verhalten mit verschiedenen Optionen ändern, die in der Manpage zu `locate` erläutert sind.

Da `locate` aus einer Datenbank liest, finden Sie möglicherweise keine Datei, die Sie kürzlich erstellt haben. Die Datenbank wird von einem Programm namens `updatedb` aktualisiert. Normalerweise wird es regelmäßig ausgeführt, aber wenn Sie root-Rechte haben und die Datenbank sofort aktualisiert werden muss, können Sie den Befehl `updatedb` jederzeit selbst ausführen.

Der Befehl `find`

`find` ist ein weiterer beliebter Befehl zur Dateisuche. Er verfolgt einen anderen Ansatz als `locate`. `find` durchsucht einen Verzeichnisbaum rekursiv, einschließlich seiner Unterverzeichnisse. `find` führt eine solche Suche bei jedem Aufruf durch, verwaltet also keine Datenbank wie `locate`. Ähnlich wie `locate` unterstützt `find` auch Wildcards und reguläre Ausdrücke.

`find` benötigt mindestens den zu druchsuchenden Pfad. Darüber hinaus können so genannte Ausdrücke hinzugefügt werden, um Filterkriterien für die anzuzeigenden Dateien anzugeben, etwa der Ausdruck `-name`, der nach Dateien mit einem bestimmten Namen sucht:

```
~$ cd Downloads
~/Downloads
$ find . -name thesis.pdf
./thesis.pdf
~/Downloads
```

```
$ find ~ -name thesis.pdf
/home/carol/Downloads/thesis.pdf
```

Der erste `find`-Befehl sucht nach der Datei im aktuellen Verzeichnis `Downloads`, während der zweite nach der Datei im Heimatverzeichnis des Benutzers sucht.

`find` ist sehr komplex, daher wird er in der Prüfung Linux Essentials nicht behandelt; er ist aber ein mächtiges und nützliches Werkzeug in der Praxis.

Geführte Übungen

1. Nutzen Sie den Befehl `man`, um herauszufinden, was die folgenden Befehle bewirken:

Befehl	**Beschreibung**
`ls`	Zeigt den Inhalt eines Verzeichnisses.
`cat`	
`cut`	
`cd`	
`cp`	
`mv`	
`mkdir`	
`touch`	
`wc`	
`passwd`	
`rm`	
`rmdir`	
`more`	
`less`	
`whereis`	
`head`	
`tail`	
`sort`	
`tr`	
`chmod`	
`grep`	

2. Öffnen Sie die Info-Seite von `ls` und finden Sie das MENU.
   - Welche Optionen haben Sie?
   - Finden Sie die Option, mit der Sie die Ausgabe nach dem Zeitpunkt der letzten Änderung

sortieren.

3. Zeigen Sie den Pfad zu den ersten 3 `README`-Dateien. Verwenden Sie den Befehl `man`, um die richtige Option für `locate` zu ermitteln.

4. Erstellen Sie eine Datei namens `test` in Ihrem Home-Verzeichnis und finden Sie deren absoluten Pfad mit dem Befehl `locate`.

5. Haben Sie sie sofort gefunden? Was mussten Sie tun, damit `locate` sie findet?

6. Suchen Sie nach der Testdatei, die Sie zuvor erstellt haben, mit dem Befehl `find`. Welche Syntax haben Sie verwendet und wie lautet der absolute Pfad ?

Offene Übungen

1. Es gibt einen Befehl in der obigen Tabelle, der keine Manpage hat. Welcher ist es und warum, glauben Sie, gibt es keine Manpage?

2. Erstellen Sie mit Hilfe der Befehle aus der obigen Tabelle den folgenden Dateibaum. Namen, die mit einem Großbuchstaben beginnen, bezeichnen Verzeichnisse, jene mit Kleinbuchstaben Dateien.

```
User
├── Documents
|   ├──Hello
|   |   ├──hey2
|   |   ├──helloa
|   |   └──ola5
|   └──World
|       └──earth9
├── Downloads
|   ├──Music
|   └──Songs
|       ├──collection1
|       └──collection2
├── Test
|   └── passa
└── test
```

3. Zeigen Sie auf dem Bildschirm das aktuelle Arbeitsverzeichnis an, einschließlich der Unterverzeichnisse.

4. Suchen Sie im Baum nach allen Dateien, die mit einer Ziffer enden.

5. Entfernen Sie den gesamten Verzeichnisbaum mit einem einzigen Befehl.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Wie Sie Hilfe erhalten
- Wie Sie den Befehl `man` verwenden
- Wie Sie in der Manpage navigieren
- Verschiedene Abschnitte der Manpage
- Wie Sie den Befehl `info` verwenden
- Wie Sie zwischen verschiedenen Knoten navigieren
- Wie Sie nach Dateien im System suchen

Befehle, die in den Übungen verwendet werden:

`man`
: Zeigt die Manpage an.

`info`
: Zeigt die Info-Seite an.

`locate`
: Durchsucht die `locate`-Datenbank nach Dateien mit dem angegebenen Namen.

`find`
: Durchsucht das Dateisystem nach Namen, die einer Reihe von Auswahlkriterien entsprechen.

`updatedb`
: Aktualisiert die `locate`-Datenbank.

Lösungen zu den geführten Übungen

1. Nutzen Sie den Befehl `man`, um herauszufinden, was die folgenden Befehle bewirken:

Befehl	**Beschreibung**
`ls`	Zeigt den Inhalts eines Verzeichnisses
`cat`	Verkettet Textdateien oder zeigt sie an
`cut`	Entfernt Abschnitte aus einer Textdatei
`cd`	Wechselt in ein anderes Verzeichnis
`cp`	Kopiert eine Datei
`mv`	Verschiebt eine Datei (kann auch zum Umbenennen verwendet werden)
`mkdir`	Erstellt ein neuen Verzeichnis
`touch`	Erstellt eine Datei oder ändert die Angabe zum Zeitpunkt der letzten Änderung einer bestehenden Datei
`wc`	Zählt die Anzahl der Wörter, Zeilen oder Bytes einer Datei
`passwd`	Ändert das Passwort eines Benutzers
`rm`	Löscht eine Datei
`rmdir`	Löscht ein Verzeichnis
`more`	Zeigt Textdateien Bildschirm für Bildschirm an
`less`	Zeigt Textdateien an, erlaubt Blättern zeilen- oder bildschirmweise
`whereis`	Zeigt den Pfad zu einem angegebenen Programm und den zugehörigen Dokumenationsdateien
`head`	Zeigt die ersten Zeilen einer Datei an
`tail`	Zeigt die letzten Zeilen einer Datei an
`sort`	Sortiert Dateien numerisch oder alphabetisch

Befehl	**Beschreibung**
`tr`	Wandelt Zeichen(folgen) in einer Datei um oder löscht sie
`chmod`	Ändert die Zugriffsrechte einer Datei
`grep`	Sucht innerhalb einer Datei

2. Öffnen Sie die Info-Seite von `ls` und finden Sie das MENU.
   - Welche Optionen haben Sie?
     - Which files are listed (Welche Dateien werden angezeigt)
     - What information is listed (Welche Informationen werden angezeigt)
     - Sorting the output (Ausgabe sortieren)
     - Details about version sort (Details zur Sortierung nach Version)
     - General output formatting (Allgemeines Ausgabeformat)
     - Formatting file timestamps (Format von Zeitstempeln)
     - Formatting the file names (Format von Dateinamen)
   - Finden Sie die Option, mit der Sie die Ausgabe nach dem Zeitpunkt der letzten Dateiänderung sortieren.

     `-t` oder `--sort=time`

3. Zeigen Sie den Pfad zu den ersten 3 `README`-Dateien. Verwenden Sie den Befehl `man`, um die richtige Option für `locate` zu ermitteln.

```
$ locate -l 3 README
/etc/alternatives/README
/etc/init.d/README
/etc/rc0.d/README
```

4. Erstellen Sie eine Datei namens `test` in Ihrem Home-Verzeichnis und finden Sie deren absoluten Pfad mit dem Befehl `locate`.

```
$ touch test
$ locate test
/home/user/test
```

5. Haben Sie sie sofort gefunden? Was mussten Sie tun, damit `locate` sie findet?

```
$ sudo updatedb
```

   Die Datei wurde eben erst erzeugt, darum existiert noch kein Eintrag in der Datenbank.

6. Suchen Sie nach der Testdatei, die Sie zuvor erstellt haben, mit dem Befehl `find`. Welche Syntax haben Sie verwendet und wie lautet der absolute Pfad ?

```
$ find ~ -name test
```

   oder

```
$ find . -name test
/home/user/test
```

Lösungen zu den offenen Übungen

1. Es gibt einen Befehl in der obigen Tabelle, der keine Manpage hat. Welcher ist es und warum, glauben Sie, gibt es keine Manpage?

   Es ist der Befehl `cd`. Er hat keine Manpage, weil es sich um einen Built-in-Befehl der Shell handelt.

2. Erstellen Sie mit Hilfe der Befehle aus der obigen Tabelle den folgenden Dateibaum. Namen, die mit einem Großbuchstaben beginnen, bezeichnen Verzeichnisse, jene mit Kleinbuchstaben Dateien.

   ```
   User
   ├── Documents
   |   ├──Hello
   |   |   ├──hey2
   |   |   ├──helloa
   |   |   └──ola5
   |   └──World
   |       └──earth9
   ├── Downloads
   |   ├──Music
   |   └──Songs
   |       ├──collection1
   |       └──collection2
   ├── Test
   |   └── passa
   └── test
   ```

   Die Lösung ist eine Kombination der Befehle `mkdir` und `touch`.

3. Zeigen Sie auf dem Bildschirm das aktuelle Arbeitsverzeichnis an, einschließlich der Unterverzeichnisse.

   ```
   $ ls -R
   ```

4. Suchen Sie im Baum nach allen Dateien, die mit einer Ziffer enden.

   ```
   $ find ~ -name "*[0-9]"
   $ locate "*[0-9]"
   ```

5. Entfernen Sie den gesamten Verzeichnisbaum mit einem einzigen Befehl.

```
$ rm -r Documents Downloads Test test
```

2.3 Verzeichnisse verwenden und Dateien auflisten

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 2.3

Gewichtung

2

Hauptwissensgebiete

- Dateien, Verzeichnisse
- Versteckte Dateien und Verzeichnisse
- Heimverzeichnisse
- Absolute und relative Pfade

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- Gängige Optionen für `ls`
- Rekursive Auflistung
- `cd`
- `.` und `..`
- `home` und `~`

2.3 Lektion 1

Einführung

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.3 Verzeichnisse verwenden und Dateien auflisten
Lektion:	1 von 2

Dateien und Verzeichnisse

Das Linux-Dateisystem ähnelt Dateisystemen anderer Betriebssysteme, insofern es *Dateien* und *Verzeichnisse* enthält. Dateien enthalten Daten wie menschenlesbaren Text, ausführbare Programme oder binäre Daten, die der Computer nutzt. Verzeichnisse dienen dazu, das Dateisystem zu organisieren.

```
$ tree

Documents
├── Mission-Statement.txt
└── Reports
    └── report2018.txt

1 directory, 2 files
```

In diesem Beispiel ist `Documents` ein Verzeichnis, das eine Datei (`Mission-Statement.txt`) und ein *Unterverzeichnis* (`Reports`) enthält, während das `Reports`-Verzeichnis wiederum eine Datei namens `report2018.txt` enthält. Das Verzeichnis `Documents` bezeichnet man als *Elternverzeichnis* des Verzeichnisses `Reports`.

TIP Steht der Befehl `tree` auf Ihrem System nicht zur Verfügung, installieren Sie ihn mit dem Paketmanager Ihrer Linux-Distribution. Schauen Sie in der Lektion über Paketverwaltung nach, wie das geht.

Datei- und Verzeichnisnamen

Datei- und Verzeichnisnamen unter Linux können Groß- und Kleinbuchstaben, Ziffern, Leerzeichen und Sonderzeichen enthalten. Da viele Sonderzeichen jedoch eine besondere Bedeutung in der Linux-Shell haben, ist es ratsam, bei der Benennung von Dateien oder Verzeichnissen keine Leerzeichen oder Sonderzeichen zu verwenden, da Leerzeichen beispielsweise das *Escape-Zeichen* `\` für die korrekte Eingabe benötigen:

```
$ cd Mission\ Statements
```

Beachten Sie den Dateinamen `report2018.txt`. Dateinamen können ein *Suffix* enthalten, das auf einen Punkt (`.`) folgt. Im Gegensatz zu Windows hat dieses Suffix unter Linux keine besondere Bedeutung; es dient dem besseren Verständnis. In unserem Beispiel zeigt `.txt` an, dass es sich um eine Klartextdatei handelt, obwohl sie technisch gesehen jede Art von Daten enthalten könnte.

Durch das Dateisystem navigieren

Den aktuellen Standort ermitteln

Da Linux-Shells wie die Bash textbasiert sind, ist es wichtig, beim Navigieren durch das Dateisystem den aktuellen Standort zu kennen, den die *Kommandozeile* oder *Eingabeaufforderung* angibt:

```
user@hostname ~/Documents/Reports $
```

Informationen zu `user` und `hostname` finden Sie in anderen Lektionen. Die Eingabeaufforderung verrät uns, dass unser aktueller Standort das Verzeichnis `Reports` ist. Dieselbe Auskunft gibt auch der Befehl `pwd` (*print working directory*), der das aktuelle *Arbeitsverzeichnis* ausgibt:

```
user@hostname ~/Documents/Reports $ pwd
```

```
/home/user/Documents/Reports
```

Die Beziehung zwischen Verzeichnissen repräsentiert der Schrägstrich oder *Slash* (`/`). Wir wissen, dass `Reports` ein Unterverzeichnis von `Documents` ist, das wiederum ein Unterverzeichnis von `user` ist, das sich in einem Verzeichnis namens `home` befindet. `home` scheint kein Elternverzeichnis zu haben, aber das stimmt nicht: Das Elternverzeichnis von `home` heißt *root* und wird durch den ersten Slash (`/`) repräsentiert.

Beachten Sie, dass sich die Ausgabe des Befehls `pwd` geringfügig von dem in der Eingabeaufforderung angegebenen Pfad unterscheidet: Anstelle von `/home/user` zeigt die Eingabeaufforderung eine Tilde (`~`). Die Tilde ist ein Sonderzeichen, das für das Heimatverzeichnis des Benutzers steht. Die nächste Lektion geht näher darauf ein.

Verzeichnisinhalt auflisten

Den Inhalt des aktuellen Verzeichnisses listet der Befehl `ls` auf:

```
user@hostname ~/Documents/Reports $ ls
report2018.txt
```

Beachten Sie, dass `ls` keine Informationen über das übergeordnete Verzeichnis liefert. `ls` zeigt standardmäßig auch keine Informationen über den Inhalt von Unterverzeichnissen an. `ls` kann nur “sehen”, was sich im aktuellen Verzeichnis befindet.

Aktuelles Verzeichnis wechseln

Die Navigation unter Linux erfolgt in erster Linie mit dem Befehl `cd` (*change directory*), er wechselt also das Verzeichnis. Mit dem Befehl `pwd` haben wir bereits ermittelt, dass unser aktuelles Verzeichnis `/home/user/Documents/Reports` ist. Wir wechseln unser aktuelles Verzeichnis, indem wir einen neuen Pfad eingeben:

```
user@hostname ~ $ cd /home/user/Documents
user@hostname ~/Documents $ pwd
/home/user/Documents
user@hostname ~/Documents $ ls
Mission-Statement.txt Reports
```

Von unserem neuen Standort aus können wir `Mission-Statement.txt` und unser Unterverzeichnis `Reports` “sehen”, aber nicht den Inhalt des Unterverzeichnisses. Zurück zu `Reports` navigieren wir wie folgt:

```
User@hostname ~/Documents $ cd Reports
user@hostname ~/Documents/Reports $ pwd
/home/user/Documents/Reports
user@hostname ~/Documents/Reports $ ls
report2018.txt
```

Wir sind jetzt wieder da, wo wir gestartet sind.

Absolute und relative Pfade

Der Befehl `pwd` gibt immer einen *absoluten Pfad* aus, d.h. der Pfad enthält jeden Schritt des Pfades, vom Ausgangspunkt des Dateisystems (`/`) bis zum aktuellen Punkt (`Reports`). Absolute Pfade beginnen immer mit einem `/`.

```
/
└── home
    └── user
        └── Documents
            └── Reports
```

Der absolute Pfad enthält alle Informationen, um von überall im Dateisystem zu `Reports` zu gelangen, mit dem Nachteil, dass es mühsam zu tippen ist.

Das zweite Beispiel (`cd Reports`) war deutlich einfacher zu tippen. Dies ist ein Beispiel für einen *relativen Pfad*. Relative Pfade sind kürzer, beschreiben den Pfad aber immer in Bezug auf die aktuelle Position. Dazu folgende Analogie: Ich besuche Sie in Ihrem Haus, und Sie sagen mir, Ihr Freund wohnt nebenan; ich werde diese Ortsangabe verstehen, weil sie relativ zu meinem aktuellen Standort ist. Wenn Sie mir diese Beschreibung aber am Telefon geben, werde ich das Haus Ihres Freundes nicht finden.

Spezielle relative Pfade

Die Linux-Shell gibt uns Mittel an die Hand, Pfadangaben zu verkürzen. Um den ersten solch speziellen Pfad kennenzulernen, geben wir den Befehl `ls` mit der Option `-a` ein. Diese ändert den Befehl `ls` so, dass *alle* Dateien und Verzeichnisse aufgelistet werden, einschließlich versteckter Dateien und Verzeichnisse:

```
user@hostname ~/Documents/Reports $ ls -a
.
..
```



```
report2018.txt
```

NOTE Werfen Sie einen Blick in die Manpage von `ls`, um zu verstehen, was `-a` hier bewirkt.

Dieser Befehl liefert zwei weitere Ergebnisse: Dies sind spezielle Pfade. Sie stehen nicht für neue Dateien oder Verzeichnisse, sondern Verzeichnisse, die Sie bereits kennen:

`.`
: Steht für den *aktuellen Standort* (in diesem Fall `Reports`).

`..`
: Steht für das *Elternverzeichnis* (in diesem Fall `Documents`).

Normalerweise ist es unnötig, den speziellen relativen Pfad für den aktuellen Standort zu nutzen. Es ist einfacher und verständlicher, `report2018.txt` zu tippen als `./report2018.txt`. Aber es gibt Einsatzzwecke für `.`, die Sie in späteren Lektionen kennenlernen werden. Im Moment konzentrieren wir uns auf den relativen Pfad für das übergeordnete Verzeichnis:

```
user@hostname ~/Documents/Reports $ cd ..
user@hostname ~/Documents $ pwd
/home/user/Documents
```

Das Beispiel von `cd` ist viel einfacher, wenn man `..` anstelle des absoluten Pfades verwendet, und wir können dieses Muster kombinieren, um sehr schnell im Dateibaum nach oben zu navigieren.

```
user@hostname ~/Documents $ cd ../..
$ pwd
/home
```

Geführte Übungen

1. Geben Sie für jeden der folgenden Pfade an, ob es sich um einen *absoluten* der *relativen* Pfad handelt:

`/home/user/Downloads`	
`../Reports`	
`/var`	
`docs`	
`/`	

2. Betrachten Sie die folgende Dateistruktur. Beachten Sie: Verzeichnisse enden mit einem Slash (`/`), wenn Sie `tree` mit der Option `-F` nutzen. Sie benötigen die entsprechende Berechtigung, um `tree` für das Root-Verzeichnis (`/`) aufzurufen. Die folgende Beispielausgabe zeigt keine vollständige Verzeichnisstruktur. Nutzen Sie sie zur Beantwortung der folgenden Fragen:

```
$ sudo tree -F /

/
├── etc/
│   ├── network/
│   │   └── interfaces
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d/
│       └── udev.conf
└── home/
    ├── lost+found/
    └── user/
        └── Documents/

12 directories, 5 files
```

   Beantworten Sie vor diesem Hintergrund die folgenden Fragen.

Ein Benutzer gibt die folgenden Befehle ein:

```
$ cd /etc/udev
$ ls -a
```

Wie lautet die Ausgabe des Befehls `ls -a`?

3. Geben Sie den jeweils kürzesten Befehl an:
   - Ihr aktueller Standort ist root (`/`). Geben Sie den Befehl an, mit dem ins Verzeichnis `lost+found` im Verzeichnis `home` gelangen (Beispiel):

     ```
     $ cd home/lost+found
     ```

   - Ihr aktueller Standort ist root (`/`). Geben Sie den Befehl an, mit dem Sie ins Verzeichnis `/etc/network/` gelangen.

   - Ihr aktueller Standort ist `/home/user/Documents/`. Geben Sie den Befehl an, mit dem Sie ins Verzeichnis `/etc/` gelangen.

   - Ihr aktueller Standort ist `/etc/systemd/system/`. Geben Sie den Befehl an, mit dem Sie ins Verzeichnis `/home/user/` gelangen.

4. Betrachten Sie die folgenden Befehle:

```
$ pwd
/etc/udev/rules.d
$ cd ../../systemd/user
$ cd ..
$ pwd
```

Wie lautet die Ausgabe des letzten `pwd` Befehls?

Offene Übungen

1. Angenommen ein Benutzer hat folgende Befehle eingegeben:

```
$ mkdir "this is a test"
$ ls
this is a test
```

   Mit welchem `cd`-Befehl könnten Sie in dieses Verzeichnis wechseln?

2. Wiederholen Sie dies, aber drücken Sie nach der Eingabe von `cd this` die TAB-Taste. Was erscheint nun in der Befehlszeile?

   Dies ist ein Beispiel für *Autocompletion*, ein wertvolles Werkzeug, das nicht nur Zeit spart, sondern auch Tippfehler vermeidet.

3. Versuchen Sie, ein Verzeichnis zu erstellen, dessen Name das Zeichen `\` enthält. Zeigen Sie den Namen des Verzeichnisses mit `ls` an und löschen Sie das Verzeichnis.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Die Grundlagen des Linux-Dateisystems
- Den Unterschied zwischen *Elternverzeichnissen* und *Unterverzeichnissen*
- Den Unterschied zwischen *absoluten* und *relativen* Dateipfaden
- Die speziellen relativen Pfade `.` und `..`
- Das Navigieren durch das Dateisystem mit `cd`
- Das Anzeigen Ihres aktuellen Standorts mit `pwd`
- Das Auflisten *aller* Dateien und Verzeichnisse mit `ls -a`

Die folgenden Befehle wurden in dieser Lektion behandelt:

`cd`
: Wechselt das aktuelle Verzeichnis.

`pwd`
: Zeigt den Pfad des aktuellen Verzeichnisses.

`ls`
: Listet den Inhalt eines Verzeichnisses auf und zeige die Eigenschaften von Dateien an.

`mkdir`
: Erstellt ein neues Verzeichnis.

`tree`
: Zeigt eine hierarchische Auflistung eines Verzeichnisbaums an.

Lösungen zu den geführten Übungen

1. Geben Sie für jeden der folgenden Pfade an, ob es sich um einen *absoluten* der *relativen* Pfad handelt:

`/home/user/Downloads`	absolut
`../Reports`	relativ
`/var`	absolut
`docs`	relativ
`/`	absolut

2. Betrachten Sie die folgende Dateistruktur. Beachten Sie: Verzeichnisse enden mit einem Slash (`/`), wenn Sie `tree` mit der Option `-F` nutzen. Sie benötigen die entsprechende Berechtigung, um `tree` für das Root-Verzeichnis (`/`) aufzurufen. Die folgende Beispielausgabe zeigt keine vollständige Verzeichnisstruktur. Nutzen Sie sie zur Beantwortung der folgenden Fragen:

```
$ sudo tree -F /

/
├── etc/
│   ├── network/
│   │   └── interfaces
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d/
│       └── udev.conf
└── home/
    ├── lost+found/
    └── user/
        └── Documents/

12 directories, 5 files
```

   Beantworten Sie vor diesem Hintergrund die folgenden Fragen.

Ein Benutzer gibt die folgenden Befehle ein:

```
$ cd /etc/udev
$ ls -a
```

Wie lautet die Ausgabe des Befehls `ls -a`?

```
. .. rules.d udev.conf
```

3. Geben Sie den jeweils kürzesten Befehl an:
   - Ihr aktueller Standort ist root (`/`). Geben Sie den Befehl an, mit dem ins Verzeichnis `lost+found` im Verzeichnis `home` gelangen (Beispiel):

     ```
     $ cd home/lost+found
     ```

   - Ihr aktueller Standort ist root (`/`). Geben Sie den Befehl an, mit dem Sie ins Verzeichnis `/etc/network/` gelangen.

     ```
     $ cd etc/network
     ```

   - Ihr aktueller Standort ist `/home/user/Documents/`. Geben Sie den Befehl an, mit dem Sie ins Verzeichnis `/etc/` gelangen.

     ```
     $ cd /etc
     ```

   - Your current location is `system`. Navigate to the directory named `user`:

     ```
     $ cd /home/user
     ```

4. Betrachten Sie die folgenden Befehle:

```
$ pwd
/etc/udev/rules.d
$ cd ../../systemd/user
$ cd ..
$ pwd
```

Wie lautet die Ausgabe des letzten `pwd` Befehls?

```
/etc/systemd
```

Lösungen zu den offenen Übungen

1. Angenommen ein Benutzer hat folgende Befehle eingegeben:

```
$ mkdir "this is a test"
$ ls
this is a test
```

   Mit welchem `cd`-Befehl könnten Sie in dieses Verzeichnis wechseln?

```
$ cd this\ is\ a\ test
```

2. Wiederholen Sie dies, aber drücken Sie nach der Eingabe von `cd this` die TAB-Taste. Was erscheint nun in der Befehlszeile?

```
$ cd this\ is\ a\ test
```

   Dies ist ein Beispiel für *Autocompletion*, ein wertvolles Werkzeug, das nicht nur Zeit spart, sondern auch Tippfehler vermeidet

3. Versuchen Sie, ein Verzeichnis zu erstellen, dessen Name das Zeichen `\` enthält. Zeigen Sie den Namen des Verzeichnisses mit `ls` an und löschen Sie das Verzeichnis.

   Sie können den Backslash entweder mit einem weiteren Backslash maskieren (`\\`) oder den gesamten Verzeichnisnamen in einfache oder doppelte Anführungszeichen setzen:

```
$ mkdir my\\dir
$ ls
'my\dir'
$ rmdir 'my\dir'
```

2.3 Lektion 2

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Objective:	2.3 Verzeichnisse verwenden und Dateien auflisten
Lektion:	2 von 2

Einführung

Das Unix Betriebssystem wurde Mitte der 1960er Jahre für Großrechner entwickelt. Viele Benutzer teilten sich solche Rechner und griffen über *Terminals* auf die Ressourcen des Systems zu. Diesem Ansatz folgen auch heutige Linux-Systeme. So sprechen wir immer noch von “Terminals” zur Eingabe von Befehlen in die Shell, und jedes Linux-System ist so organisiert, dass viele Benutzer auf einem einzigen System einfach anzulegen sind.

Heimatverzeichnisse

Dies ist ein Beispiel für ein normales Dateisystem unter Linux:

```
$ tree -L 1 /
/
├── bin
├── boot
├── cdrom
├── dev
```

```
├── etc
├── home
├── lib
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

Die meisten dieser Verzeichnisse sind auf allen Linux-Systemen zu finden: Von Servern über Supercomputer bis hin zu winzigen eingebetteten Systemen kann ein erfahrener Linux-Benutzer davon ausgehen, dass er den Befehl `ls` in `/bin` findet, dass er die Systemkonfiguration durch Anpassung von Dateien in `/etc` ändert und dass er Systemprotokolle in `/var` liest. Die Positionen dieser Dateien und Verzeichnisse definiert der *Filesystem Hierarchy Standard* (FHS), den eine spätere Lektion behandelt. Sie werden im täglichen Umgang mit Linux mehr über die Inhalte dieser Verzeichnisse lernen, aber fürs erste sollten Sie sich merken:

- Änderungen, die Sie im Root-Dateisystem vornehmen, betreffen alle Benutzer und
- Änderungen von Dateien im Root-Dateisystem erfordern Administratorrechte.

Das bedeutet, dass es normalen Nutzern untersagt ist, diese Dateien zu ändern oder sogar zu lesen; wir werden das Thema Berechtigungen in einem späteren Abschnitt behandeln.

Nun konzentrieren wir uns auf das Verzeichnis `/home`, das an dieser Stelle bereits etwas vertraut sein sollte:

```
$ tree -L 1 /home
/home
├── user
├── michael
└── lara
```

Unser Beispielsystem hat drei normale Benutzer, und jeder unserer Benutzer hat seinen eigenen Bereich, in dem er Dateien und Verzeichnisse erstellen und ändern kann, ohne seine Nachbarn zu beeinträchtigen. In der vorherigen Lektion haben wir mit der folgenden Dateistruktur gearbeitet:

```
$ tree /home/user
user
└── Documents
    ├── Mission-Statement
    └── Reports
        └── report2018.txt
```

Tatsächlich sieht ein echtes Dateisystem aber eher so aus:

```
$ tree /home
/home
├── user
│   └── Documents
│       ├── Mission-Statement
│       └── Reports
│           └── report2018.txt
├── michael
│   ├── Documents
│   │   └── presentation-for-clients.odp
│   └── Music
```

...dasselbe gilt für `lara`.

Unter Linux ist `/home` ähnlich wie ein Mehrfamilienhaus: Viele Nutzer haben ihren eigenen, abgetrennten Bereich. Versorgung und Wartung des Gebäudes selbst liegt hingegen in der Verantwortung des Hausverwalters namens *root*.

Der besondere relative Pfad für das Heimatverzeichnis

Wenn Sie eine neue Terminalsitzung unter Linux starten, sehen Sie eine ähnliche Eingabeaufforderung:

```
user@hostname ~ $
```

Die Tilde (`~`) repräsentiert unser *Heimatverzeichnis*. Wenn Sie den Befehl `ls` ausführen, sehen Sie einige bekannte Ausgaben:

```
$ cd ~
$ ls
```

```
Documents
```

Vergleichen Sie das zum besseren Verständnis mit dem obigen Dateisystem.

Was wir nun also über Linux wissen: Es ist einem Mehrfamilienhaus ähnlich, in dem viele Benutzer in `/home` wohnen. Der Bereich des Benutzers `user` wird sich folglich von dem des Benutzers `michael` unterscheiden. Das werden wir mit dem Befehl `su` (“switch user”) zeigen, der den Benutzer wechselt.

```
user@hostname ~ $ pwd
/home/user
user@hostname ~ $ su - michael
Password:
michael@hostname ~ $ pwd
/home/michael
```

Die Bedeutung von `~` ändert sich je nach Benutzer: Für `michael` ist der absolute Pfad von `~` `/home/michael`, für `lara` ist er `/home/lara` und so weiter.

Relative-to-Home-Pfade

Die Verwendung von `~` in Befehlen ist sehr praktisch, vorausgesetzt, Sie wechseln nicht den Benutzer. Betrachten wir das folgende Beispiel für `user`, der eine neue Sitzung begonnen hat:

```
$ ls
Documents
$ cd Documents
$ ls
Mission-Statement
Reports
$ cd Reports
$ ls
report2018.txt
$ cd ~
$ ls
Documents
```

Beachten Sie, dass Benutzer eine neue Sitzung stets in ihrem Heimatverzeichnis beginnen. In diesem Beispiel ist `user` in das Unterverzeichnis `Documents/Reports` gereist. Mit dem Befehl `cd ~` ist er zum Ausgangspunkt zurückgekehrt; dasselbe hätte er mit dem Befehl `cd` ohne Argumente

erreicht:

```
$ cd Documents/Reports
$ pwd
/home/user/Documents/Reports
$ cd
$ pwd
/home/user
```

Eine letzte Anmerkung: Wir können die Home-Verzeichnisse *anderer* Benutzer angeben, indem wir den Benutzernamen direkt nach der Tilde angeben, zum Beispiel:

```
$ ls ~michael
Documents
Music
```

Beachten Sie, dass das nur funktioniert, wenn `michael` uns die Erlaubnis erteilt hat, den Inhalt seines Home-Verzeichnisses zu sehen.

Nehmen wir an, `michael` möchte die Datei `report2018.txt` im Home-Verzeichnis von `user` lesen. Er hat die Erlaubnis dazu und kann den Befehl `less` verwenden.

```
$ less ~user/Documents/Reports/report2018.txt
```

Jeder Dateipfad, der das Zeichen `~` enthält, wird als *Relative-to-Home*-Pfad bezeichnet.

Versteckte Dateien und Verzeichnisse

In der vorangegangenen Lektion haben wir die Option `-a` für den Befehl `ls` und mit `ls -a` die beiden speziellen relativen Pfade `.` und `..` eingeführt. Die Option `-a` listet *alle* Dateien und Verzeichnisse auf, einschließlich *versteckter* Dateien und Verzeichnisse.

```
$ ls -a ~
.
..
.bash_history
.bash_logout
.bash-profile
.bashrc
```

```
Documents
```

Versteckte Dateien und Verzeichnisse beginnen immer mit einem Punkt (`.`). Standardmäßig enthält das Heimatverzeichnis eines Benutzers viele versteckte Dateien. Darin sind häufig benutzerspezifische Konfigurationseinstellungen abgelegt, und sie sollten nur von einem erfahrenen Benutzer geändert werden.

Die Long-list-Option

Der Befehl `ls` hat viele Optionen, um sein Verhalten zu ändern. Schauen wir uns eine der meistgenutzten an:

```
$ ls -l
-rw-r--r-- 1 user staff      3606 Jan 13  2017 report2018.txt
```

`-l` erstellt eine "long list" ("lange Liste"). Dateien und Verzeichnisse belegen in der Ausgabe jeweils eine Zeile, und es werden zusätzliche Informationen über jede Datei und jedes Verzeichnis angezeigt.

`-rw-r--r--`

Dateityp und Berechtigungen der Datei. Beachten Sie, dass eine normale Datei mit Bindestrich beginnt und ein Verzeichnis mit `d`.

`1`

Anzahl der Links zur Datei.

`user staff`

Gibt den Eigentümer der Datei an, in diesem Fall also `user`. Zudem ist die Datei der Gruppe `staff` zugeordnet.

`3606`

Größe der Datei in Bytes.

`Jan 13 2017`

Zeitstempel der letzten Änderung an der Datei.

`report2018.txt`

Name der Datei.

Themen wie Eigentümer, Berechtigungen und Links werden in späteren Lektionen behandelt.

Wie Sie sehen, ist die -List-Variante von `ls` oft dem Standardaufruf vorzuziehen.

Weitere `ls`-Optionen

Nachfolgend finden Sie einige Varianten, wie der Befehl `ls` am häufigsten verwendet wird. Wie Sie sehen, lassen sich mehrere Optionen für die gewünschte Ausgabe kombinieren.

`ls -lh`

Die Kombination von “langer Liste” (“long list”) mit “menschenlesbaren” (“human readable”) Dateigrößen gibt uns nützliche Suffixe wie `M` für Megabyte oder `K` für Kilobyte.

`ls -d */`

Die Option `-d` listet Verzeichnisse auf, aber nicht deren Inhalt. Kombiniert mit `*/` zeigt sie nur Unterverzeichnisse und keine Dateien.

`ls -lt`

Kombiniert “lange Liste” mit der Option, nach “Zeitpunkt der letzten Änderung” zu sortieren. Dateien mit den letzten Änderungen stehen oben, Dateien mit den ältesten Änderungen unten, wobei die Reihenfolge auch umgekehrt werden kann.

`ls -lrt`

Kombiniert “lange Liste” mit “Zeitpunkt der letzten Änderung” und `-r` für “umgekehrte Sortierung” (“reverse order”), so dass Dateien mit den letzten Änderungen am Ende der Liste stehen. Neben der Sortierung nach “Zeitpunkt der letzten Änderung” sind auch “Zeitpunkt des letzten Zugriffs” oder nach “Zeitpunkt der letzten Statusänderung” möglich.

`ls -lX`

Kombiniert “lange Liste” mit der Sortierung nach Dateiendungen (“eXtension”), um z.B. alle Dateien, die mit `.txt` oder mit `.jpg` enden, zusammenzufassen.

`ls -S`

`-S` sortiert nach Dateigröße, so wie `-t` nach Zeit oder `-X` nach Dateiendung, wobei die größten Dateien an erster Stelle stehen und die kleinsten zuletzt. Inhalte von Unterverzeichnissen sind übrigens von der Sortierung ausgenommen.

`ls -R`

Die Option `-R` bewirkt für den Befehl `ls`, dass er eine *rekursive* Liste ausgibt. Was bedeutet das?

Rekursion in Bash

Rekursion bezeichnet eine Situation, in der “etwas in sich selbst definiert” ist. Rekursion ist ein sehr wichtiger Begriff in der Informatik, aber hier ist seine Bedeutung viel einfacher. Betrachten wir unser Beispiel von vorhin:

```
$ ls ~
Documents
```

Wir wissen bereits, dass `user` ein Home-Verzeichnis hat und dass es in diesem Verzeichnis ein Unterverzeichnis gibt. `ls` hat uns bisher nur die Dateien und Unterverzeichnisse eines Ortes ausgegeben, kann uns aber den Inhalt dieser Unterverzeichnisse nicht anzeigen. In diesen Lektionen haben wir den Befehl `tree` verwendet, um den Inhalt vieler Verzeichnisse anzuzeigen. `tree` ist leider kein Standardwerkzeug von Linux und steht daher nicht immer zur Verfügung. Vergleichen Sie die Ausgabe von `tree` mit der von `ls -R` in den folgenden Beispielen:

```
$ tree /home/user
user
└── Documents
    ├── Mission-Statement
    └── Reports
        └── report2018.txt

$ ls -R ~
/home/user/:
Documents

/home/user/Documents:
Mission-Statement
Reports

/home/user/Documents/Reports:
report2018.txt
```

Wie Sie sehen, erhalten wir mit der Rekursiv-Option eine viel längere Liste von Dateien. Tatsächlich ist es so, als würden wir den Befehl `ls` im Heimatverzeichnis von `user` aufrufen und auf ein Unterverzeichnis treffen; daraufhin gehen wir in dieses Unterverzeichnis und führen den Befehl `ls` dort erneut aus. Wir sehen die Datei `Mission-Statement` und ein weiteres Unterverzeichnis namens `Reports`. Wir gehen in das Unterverzeichnis und haben den Befehl `ls` erneut ausgeführt. Im Grunde sagt der Befehl `ls -R` der Bash: “Führe `ls` hier aus und wiederhole

den Befehl in jedem Unterverzeichnis, das Du findest."

Rekursion ist besonders wichtig bei Änderungen von Dateien wie dem Kopieren oder Entfernen von Verzeichnissen. Wenn Sie etwa das Unterverzeichnis `Documents` kopieren möchten, müssen Sie eine rekursive Kopie vornehmen, um den Befehl auf alle Unterverzeichnisse auszuweiten.

Geführte Übungen

1. Nutzen Sie die folgende Dateistruktur, um die folgenden drei Fragen zu beantworten:

```
/
├── etc/
│   ├── network/
│   │   └── interfaces/
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d
│       └── udev.conf
└── home/
    ├── lost+found/
    ├── user/
    │   └── Documents/
    └── michael/
        └── Music/
```

   - Welcher Befehl wechselt ins Verzeichnis `network` unabhängig vom aktuellen Standort?

   - Welchen Befehl kann `user` eingeben, um von `/etc/udev` ins Verzeichnis `Documents` zu wechseln? Geben Sie den kürzestmöglichen Pfad an.

   - Welchen Befehl kann `user` eingeben, um in das Verzeichnis `music` des Benutzers `michael` zu wechseln? Nutzen Sie den kürzestmöglichen Pfad.

2. Betrachten Sie die folgende Ausgabe von `ls -lh`, um die nächsten beiden Fragen zu beantworten. Beachten Sie, dass Verzeichnisse mit einem `d` am Zeilenanfang gekennzeichnet sind.

```
drwxrwxrwx  5 eric eric  4.0K Apr 26  2011 China/
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0066.jpg
```

```
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0067.jpg
-rwxrwxrwx  1 eric eric  1.6M Jul 18  2011 img_0074.jpg
-rwxrwxrwx  1 eric eric  1.8M Jul 18  2011 img_0075.jpg
-rwxrwxrwx  1 eric eric   46K Jul 18  2011 scary.jpg
-rwxrwxrwx  1 eric eric  469K Jan 29  2018 Screenshot from 2017-08-13 21-22-24.png
-rwxrwxrwx  1 eric eric  498K Jan 29  2018 Screenshot from 2017-08-14 21-18-07.png
-rwxrwxrwx  1 eric eric  211K Jan 29  2018 Screenshot from 2018-01-06 23-29-30.png
-rwxrwxrwx  1 eric eric  150K Jul 18  2011 tobermory.jpg
drwxrwxrwx  6 eric eric  4.0K Apr 26  2011 Tokyo/
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 081.jpg
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 085.jpg
-rwxrwxrwx  1 eric eric  944K Jul 18  2011 Toronto 152.jpg
-rwxrwxrwx  1 eric eric  728K Jul 18  2011 Toronto 173.jpg
drwxrwxrwx  2 eric eric  4.0K Jun  5  2016 Wallpapers/
```

- Welche Datei steht zu Beginn, wenn Sie den Befehl `ls -lrS` ausführen?

- Bitte beschreiben Sie, welche Ausgabe Sie von dem Befehl `ls -ad */` erwarten.

Offene Übungen

1. Führen Sie den Befehl `ls -lh` in einem Verzeichnis aus, das Unterverzeichnisse enthält. Beachten Sie die angezeigte Größe dieser Verzeichnisse. Scheint Ihnen die Dateigröße korrekt? Entspricht sie dem Inhalt aller Dateien in diesem Verzeichnis?

2. Hier ein neuer Befehl zum Ausprobieren: `du -h`. Führen Sie diesen Befehl aus und beschreiben Sie dessen Ausgabe.

3. Auf vielen Linux-Systemen können Sie `ll` eingeben und erhalten die gleiche Ausgabe wie bei `ls -l`. Beachten Sie jedoch, dass `ll` *kein* Befehl ist. `man ll` wird beispielsweise darauf hinweisen, dass keine entsprechende Manpage existiert. Es ist ein Beispiel für einen *Alias*. Inwiefern können Aliase nützlich sein?

Zusammenfassung

In dieser Lektion haben Sie gelernt, dass

- jeder Linux-Benutzer ein Heimatverzeichnis hat,
- das Heimatverzeichnis des aktuellen Benutzers über `~` zu erreichen ist,
- jeder Dateipfad, der `~` verwendet, als *Relative-to-home-Pfad* bezeichnet wird.

Sie haben zudem einige der meistgenutzten Optionen für `ls` kennengelernt:

`-a` (all)
: Gibt alle Dateien/Verzeichnisse aus, einschließlich der versteckten.

`-d` (directories)
: Gibt alle Verzeichnisse aus, nicht deren Inhalt.

`-h` (human readable)
: Gibt Dateigrößen in einem menschenlesbaren Format aus.

`-l` (long list)
: Liefert zusätzliche Details mit einer Datei/einem Verzeichnis pro Zeile.

`-r` (reverse)
: Kehrt die Reihenfolge einer Sortierung um.

`-R` (recursive)
: Listet jede Datei, einschließlich der Dateien in allen Unterverzeichnissen.

`-S` (size)
: Sortiert nach Dateigröße.

`-t` (time)
: Sortiert nach dem Zeitpunkt der letzten Änderung.

`X` (eXtension)
: Sortiert nach Dateiendung.

Lösungen zu den geführten Übungen

1. Nutzen Sie die folgende Dateistruktur, um die folgenden drei Fragen zu beantworten:

```
/
├── etc/
│   ├── network/
│   │   └── interfaces/
│   ├── systemd/
│   │   ├── resolved.conf
│   │   ├── system/
│   │   ├── system.conf
│   │   ├── user/
│   │   └── user.conf
│   └── udev/
│       ├── rules.d
│       └── udev.conf
└── home/
    ├── lost+found/
    ├── user/
    │   └── Documents/
    └── michael/
        └── Music/
```

   - Welcher Befehl wechselt ins Verzeichnis `network` unabhängig vom aktuellen Standort?

     ```
     cd /etc/network
     ```

   - Welchen Befehl kann `user` eingeben, um von `/etc/udev` ins Verzeichnis `Documents` zu wechseln? Geben Sie den kürzestmöglichen Pfad an.

     ```
     cd ~/Documents
     ```

   - Welchen Befehl kann `user` eingeben, um in das Verzeichnis `music` des Benutzers `michael` zu wechseln? Nutzen Sie den kürzestmöglichen Pfad.

     ```
     cd ~michael/Music
     ```

2. Betrachten Sie die folgende Ausgabe von `ls -lh`, um die nächsten beiden Fragen zu

beantworten. Beachten Sie, dass Verzeichnisse mit einem `d` am Zeilenanfang gekennzeichnet sind.

```
drwxrwxrwx  5 eric eric  4.0K Apr 26  2011 China/
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0066.jpg
-rwxrwxrwx  1 eric eric  1.5M Jul 18  2011 img_0067.jpg
-rwxrwxrwx  1 eric eric  1.6M Jul 18  2011 img_0074.jpg
-rwxrwxrwx  1 eric eric  1.8M Jul 18  2011 img_0075.jpg
-rwxrwxrwx  1 eric eric   46K Jul 18  2011 scary.jpg
-rwxrwxrwx  1 eric eric  469K Jan 29  2018 Screenshot from 2017-08-13 21-22-24.png
-rwxrwxrwx  1 eric eric  498K Jan 29  2018 Screenshot from 2017-08-14 21-18-07.png
-rwxrwxrwx  1 eric eric  211K Jan 29  2018 Screenshot from 2018-01-06 23-29-30.png
-rwxrwxrwx  1 eric eric  150K Jul 18  2011 tobermory.jpg
drwxrwxrwx  6 eric eric  4.0K Apr 26  2011 Tokyo/
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 081.jpg
-rwxrwxrwx  1 eric eric  1.4M Jul 18  2011 Toronto 085.jpg
-rwxrwxrwx  1 eric eric  944K Jul 18  2011 Toronto 152.jpg
-rwxrwxrwx  1 eric eric  728K Jul 18  2011 Toronto 173.jpg
drwxrwxrwx  2 eric eric  4.0K Jun  5  2016 Wallpapers/
```

- Welche Datei steht zu Beginn, wenn Sie den Befehl `ls -lrS` ausführen?

  Die drei Verzeichnisse sind alle 4.0K groß, was der kleinstmöglichen Größe entspricht. `ls` wird die Verzeichnisse dann standardmäßig alphabetisch sortieren. Die richtige Antwort ist die Datei `scary.jpg`.

- Bitte beschreiben Sie, welche Ausgabe Sie von dem Befehl `ls -ad */` erwarten.

  Der Befehl wird alle Unterverzeichnisse einschließlich versteckter Unterverzeichnisse ausgeben.

Lösungen zu den offenen Übungen

1. Führen Sie den Befehl `ls -lh` in einem Verzeichnis aus, das Unterverzeichnisse enthält. Beachten Sie die angezeigte Größe dieser Verzeichnisse. Scheint Ihnen die Dateigröße korrekt? Entspricht sie dem Inhalt aller Dateien in diesem Verzeichnis?

   Nein, das tut sie nicht. Jedes Verzeichnis hat eine angezeigte Größe von 4096 Bytes, da Verzeichnisse hier eine Abstraktion sind: Sie existieren nicht als Baumstruktur auf der Festplatte. Ist ein Verzeichnis aufgelistet, so ist es ein *Link* zu einer Liste von Dateien. Die Dateigröße solcher Links beträgt 4096 Bytes.

2. Hier ein neuer Befehl zum Ausprobieren: `du -h`. Führen Sie diesen Befehl aus und beschreiben Sie dessen Ausgabe.

   Der Befehl `du` erzeugt eine Liste aller Dateien und Verzeichnisse und gibt deren jeweilige Größe an. `du -s` zeigt beispielsweise die Dateigröße aller Dateien, Verzeichnisse und Unterverzeichnisse für einen bestimmten Ort an.

3. Auf vielen Linux-Systemen können Sie `ll` eingeben und erhalten die gleiche Ausgabe wie bei `ls -l`. Beachten Sie jedoch, dass `ll` *kein* Befehl ist. `man ll` wird beispielsweise darauf hinweisen, dass keine entsprechende Manpage existiert. Es ist ein Beispiel für einen *Alias*. Inwiefern können Aliase nützlich sein?

   `ll` ist ein *Alias* von `ls -l`. In Bash dienen Aliase dazu, häufig verwendete Befehle zu vereinfachen. `ll` ist unter Linux häufig vordefiniert.

2.4 Erstellen, Verschieben und Löschen von Dateien

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 2.4

Gewichtung

2

Hauptwissensgebiete

- Dateien und Verzeichnisse
- Groß- und Kleinschreibung
- Einfaches Globbing

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

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

2.4 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.4 Erstellen, Verschieben und Löschen von Dateien
Lektion:	1 von 1

Einführung

Diese Lektion behandelt die Verwaltung von Dateien und Verzeichnissen unter Linux mit Hilfe von Befehlszeilenwerkzeugen.

Eine Datei ist eine Sammlung von Daten mit einem Namen und einer Reihe von Attributen. Wenn Sie beispielsweise Fotos von Ihrem Handy auf einen Computer übertragen und ihnen beschreibende Namen geben, haben Sie eine Reihe von Bilddateien auf Ihrem Computer. Diese Dateien haben Attribute wie die Zeit des letzten Zugriffs auf die Datei oder den Zeitpunkt der letzten Änderung.

Ein Verzeichnis ist eine spezielle Art von Datei zum Organisieren von Dateien. Stellen Sie sich Verzeichnisse als Aktenordner vor, in denen man Dokumente aufbewahrt. Aber anders als bei Aktenordnern aus Pappe können Sie in einem Verzeichnis weitere Verzeichnisse ablegen.

Die Befehlszeile ist der effektivste Weg, um Dateien auf einem Linux-System zu verwalten. Die Shell- und Kommandozeilen-Tools verfügen über Funktionen, die die Arbeit auf der Kommandozeile schneller und einfacher machen als ein grafischer Dateimanager.

In diesem Abschnitt verwenden Sie die Befehle `ls`, `mv`, `cp`, `pwd`, `find`, `touch`, `rm`, `rmdir`, `echo`, `cat` und `mkdir` zur Verwaltung und Organisation von Dateien und Verzeichnissen.

Groß-/Kleinschreibung beachten

Im Gegensatz zu Microsoft Windows ist auf Linux-Systemen bei Datei- und Verzeichnisnamen Groß-/Kleinschreibung zu unterscheiden, d.h. `/etc/` und `/ETC/` bezeichnen unterschiedliche Verzeichnisse:

```
$ cd /
$ ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
$ cd ETC
bash: cd: ETC: No such file or directory
$ pwd
/
$ cd etc
$ pwd
/etc
```

Das `pwd` zeigt Ihnen das Verzeichnis, in dem Sie sich gerade befinden. Wie Sie sehen, hat der Wechsel zu `/ETC` nicht funktioniert, da es kein solches Verzeichnis gibt. Der Wechsel in das existierende Verzeichnis `/etc` ist hingegen gelungen.

Verzeichnisse erstellen

Der Befehl `mkdir` wird verwendet, um Verzeichnisse zu erstellen.

Wir wollen nun ein neues Verzeichnis in unserem Heimatverzeichnis erstellen:

```
$ cd ~
$ pwd
/home/user
$ ls
Desktop  Documents  Downloads
$ mkdir linux_essentials-2.4
$ ls
Desktop  Documents  Downloads  linux_essentials-2.4
$ cd linux_essentials-2.4
$ pwd
```

```
/home/emma/linux_essentials-2.4
```

In dieser Lektion führen wir sämtliche Befehle in diesem Verzeichnis oder in einem seiner Unterverzeichnisse aus.

Um von jeder anderen Position in Ihrem Dateisystem einfach in das Lektionsverzeichnis zurückzukehren, können Sie den folgenden Befehl verwenden:

```
$ cd ~/linux_essentials-2.4
```

Die Shell interpretiert das Zeichen `~` als Ihr Home-Verzeichnis.

Erstellen Sie im Lektionsverzeichnis weitere Verzeichnisse, die wir für die Übungen verwenden werden. Setzen Sie alle Verzeichnisnamen, getrennt durch Leerzeichen, hinter `mkdir`:

```
$ mkdir creating moving copying/files copying/directories deleting/directories
deleting/files globs
mkdir: cannot create directory ‘copying/files’: No such file or directory
mkdir: cannot create directory ‘copying/directories’: No such file or directory
mkdir: cannot create directory ‘deleting/directories’: No such file or directory
mkdir: cannot create directory ‘deleting/files’: No such file or directory
$ ls
creating  globs  moving
```

Beachten Sie die Fehlermeldung, dass nur `moving`, `globs` und `creating` erstellt wurden. Die Verzeichnisse `copying` und `deleting` existieren noch nicht. `mkdir` erstellt standardmäßig kein Verzeichnis innerhalb eines Verzeichnisses, das nicht bereits existiert. Die Option `-p` oder `--parents` weist `mkdir` an, übergeordnete Verzeichnisse zu erstellen, wenn sie nicht vorhanden sind. Probieren Sie den gleichen Befehl `mkdir` mit der Option `-p`:

```
$ mkdir -p creating moving copying/files copying/directories deleting/directories
deleting/files globs
```

Jetzt bekommen Sie keine Fehlermeldungen mehr. Lassen Sie uns sehen, welche Verzeichnisse jetzt existieren:

```
$ find
.
./creating
```

```
./moving
./globs
./copying
./copying/files
./copying/directories
./deleting
./deleting/directories
./deleting/files
```

Das Programm `find` wird normalerweise verwendet, um nach Dateien und Verzeichnissen zu suchen, aber ohne Optionen zeigt es eine Liste aller Dateien, Verzeichnisse und Unterverzeichnisse des aktuellen Verzeichnisses.

TIP

Wenn Sie den Inhalt eines Verzeichnisses mit `ls` auflisten, sind die Optionen `-t` und `-r` besonders praktisch: Sie sortieren die Ausgabe nach Zeit (`-t`) und kehren die Sortierreihenfolge um (`-r`); in diesem Fall stehen die neuesten Dateien am Ende der Ausgabe.

Dateien erstellen

Normalerweise werden Dateien von den Programmen erstellt, die mit den in den Dateien gespeicherten Daten arbeiten. Eine leere Datei erstellen Sie mit dem Befehl `touch`. Wenn Sie `touch` auf einer bestehenden Datei ausführen, ändert sich der Inhalt der Datei nicht, aber der Zeitstempel der Dateiänderung wird aktualisiert.

Führen Sie den folgenden Befehl aus, um einige Dateien für die Lektion zum Thema “Globbing” zu erstellen:

```
$ touch globs/question1 globs/question2012 globs/question23 globs/question13
globs/question14
$ touch globs/star10 globs/star1100 globs/star2002 globs/star2013
```

Lassen Sie uns überprüfen, ob alle Dateien im Verzeichnis `globs` vorhanden sind:

```
$ cd globs
$ ls
question1   question14    question23  star1100  star2013
question13  question2012  star10      star2002
```

`touch` hat also die Dateien erstellt. Sie können den Inhalt einer Textdatei mit dem Befehl `cat`

ansehen und ihn an einer der Dateien ausprobieren, die Sie gerade erstellt haben:

```
$ cat question14
```

Da `touch` leere Dateien erzeugt, sollten Sie keine Ausgabe erhalten. Sie können `echo` mit `>` verwenden, um einfache Textdateien zu erstellen:

```
$ echo hello > question15
$ cat question15
hello
```

`echo` zeigt Text auf der Kommandozeile an. Das Zeichen `>` weist die Shell an, die Ausgabe eines Befehls in die angegebene Datei (statt ins Terminal) zu schreiben, was dazu führt, dass die Ausgabe von `echo`, in diesem Fall `hello`, in die Datei `question15` geschrieben wird. Das ist nicht spezifisch für `echo`, das geht mit jedem anderen Befehl.

WARNING Vorsicht bei der Verwendung von `>`! Wenn die genannte Datei bereits existiert, wird sie überschrieben!

Umbenennen von Dateien

Dateien werden mit dem Befehl `mv` verschoben und umbenannt.

Stellen Sie Ihr Arbeitsverzeichnis auf das Verzeichnis `moving` ein:

```
$ cd ~/linux_essentials-2.4/moving
```

Erstellen Sie einige Dateien zum Üben. Mit den notwendigen Befehlen sollten Sie bereits vertraut sein:

```
$ touch file1 file22
$ echo file3 > file3
$ echo file4 > file4
$ ls
file1  file22  file3  file4
```

Angenommen `file22` ist ein Tippfehler und sollte `file2` heißen. Korrigieren Sie es mit dem Befehl `mv`. Beim Umbenennen einer Datei ist das erste Argument der aktuelle Name, das zweite der neue Name:

```
$ mv file22 file2
$ ls
file1  file2  file3  file4
```

Seien Sie vorsichtig mit dem Befehl `mv`: Wenn Sie eine Datei in eine bereits bestehende Datei umbenennen, wird diese überschrieben. Testen wir dies mit `file3` und `file4`:

```
$ cat file3
file3
$ cat file4
file4
$ mv file4 file3
$ cat file3
file4
$ ls
file1  file2  file3
```

Beachten Sie, dass der Inhalt von `file3` jetzt `file4` ist. Verwenden Sie die Option `-i`, um `mv` anzuweisen, vor dem Überschreiben einer bestehenden Datei eine Bestätigung anzufordern:

```
$ touch file4 file5
$ mv -i file4 file3
mv: overwrite ‘file3’? y
```

Verschieben von Dateien

Dateien werden mit dem Befehl `mv` von einem Verzeichnis in ein anderes verschoben.

Erstellen Sie ein paar Verzeichnisse, in die Sie Dateien verschieben können:

```
$ cd ~/linux_essentials-2.4/moving
$ mkdir dir1 dir2
$ ls
dir1  dir2  file1  file2  file3  file5
```

Verschieben Sie `file1` in `dir1`:

```
$ mv file1 dir1
$ ls
```

```
dir1  dir2  file2  file3  file5
$ ls dir1
file1
```

Beachten Sie, dass das letzte Argument von `mv` das Zielverzeichnis ist. Wenn das letzte Argument von `mv` ein Verzeichnis ist, werden Dateien in dieses Verzeichnis verschoben, wobei mehrere Dateien in einem einzigen `mv` Befehl angegeben werden können:

```
$ mv file2 file3 dir2
$ ls
dir1  dir2  file5
$ ls dir2
file2  file3
```

Es ist auch möglich, mit `mv` Verzeichnisse zu verschieben und umzubenennen. Benennen Sie `dir1` in `dir3` um:

```
$ ls
dir1  dir2  file5
$ ls dir1
file1
$ mv dir1 dir3
$ ls
dir2  dir3  file5
$ ls dir3
file1
```

Löschen von Dateien und Verzeichnissen

Der Befehl `rm` löscht Dateien und Verzeichnisse, während der Befehl `rmdir` nur Verzeichnisse löschen kann. Räumen wir das Verzeichnis `moving` durch Löschen von `file5` auf:

```
$ cd ~/linux_essentials-2.4/moving
$ ls
dir2  dir3  file5
$ rmdir file5
rmdir: failed to remove ‘file5’: Not a directory
$ rm file5
$ ls
dir2  dir3
```

Standardmäßig kann `rmdir` nur leere Verzeichnisse löschen, darum mussten wir `rm` verwenden, um eine normale Datei zu löschen. Versuchen Sie, das Verzeichnis `deleting` zu löschen:

```
$ cd ~/linux_essentials-2.4/
$ ls
copying  creating  deleting  globs  moving
$ rmdir deleting
rmdir: failed to remove ‘deleting’: Directory not empty
$ ls -l deleting
total 0
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 directories
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 files
```

Standardmäßig weigert sich `rmdir`, ein Verzeichnis zu löschen, das nicht leer ist. Verwenden Sie `rmdir`, um eines der leeren Unterverzeichnisse im Verzeichnis `deleting` zu entfernen:

```
$ ls -a deleting/files
.  ..
$ rmdir deleting/files
$ ls -l deleting
directories
```

Das Löschen einer großen Anzahl von Dateien oder tiefer Verzeichnisstrukturen mit vielen Unterverzeichnissen mag aufwändig scheinen, ist aber eigentlich einfach. `rm` funktioniert standardmäßig nur bei normalen Dateien. Die Option `-r` wird verwendet, um dieses Verhalten zu überschreiben. Aber Vorsicht, `rm -r` ist brandgefährlich! Mit der Option `-r` löscht `rm` nicht nur alle Verzeichnisse, sondern alles in diesem Verzeichnis, einschließlich der Unterverzeichnisse und deren Inhalte. Sehen Sie selbst, wie `rm -r` funktioniert:

```
$ ls
copying  creating  deleting  globs  moving
$ rm deleting
rm: cannot remove ‘deleting’: Is a directory
$ ls -l deleting
total 0
drwxrwxr-x. 2 emma emma 6 Mar 26 14:58 directories
$ rm -r deleting
$ ls
copying  creating  globs  moving
```

Beachten Sie, wie `deleting` verschwunden ist, obwohl es nicht leer war. Wie `mv` hat `rm` eine `-i` Option, um vor der Ausführung eine Bestätigung anzufordern. Nutzen Sie `rm -ri`, um Verzeichnisse aus `moving` zu entfernen, die nicht mehr benötigt werden:

```
$ find
.
./creating
./moving
./moving/dir2
./moving/dir2/file2
./moving/dir2/file3
./moving/dir3
./moving/dir3/file1
./globs
./globs/question1
./globs/question2012
./globs/question23
./globs/question13
./globs/question14
./globs/star10
./globs/star1100
./globs/star2002
./globs/star2013
./globs/question15
./copying
./copying/files
./copying/directories
$ rm -ri moving
rm: descend into directory ‘moving’? y
rm: descend into directory ‘moving/dir2’? y
rm: remove regular empty file ‘moving/dir2/file2’? y
rm: remove regular empty file ‘moving/dir2/file3’? y
rm: remove directory ‘moving/dir2’? y
rm: descend into directory ‘moving/dir3’? y
rm: remove regular empty file ‘moving/dir3/file1’? y
rm: remove directory ‘moving/dir3’? y
rm: remove directory ‘moving’? y
```

Kopieren von Dateien und Verzeichnissen

Mit dem Befehl `cp` werden Dateien und Verzeichnisse kopiert. Kopieren Sie einige Dateien in das Verzeichnis `copying`:


```
$ cd ~/linux_essentials-2.4/copying
$ ls
directories  files
$ cp /etc/nsswitch.conf files/nsswitch.conf
$ cp /etc/issue /etc/hostname files
```

Wenn das letzte Argument ein Verzeichnis ist, erstellt `cp` eine Kopie der vorherigen Argumente innerhalb des Verzeichnisses. Wie bei `mv` kann man mehrere Dateien auf einmal angeben, solange das Ziel ein Verzeichnis ist.

Wenn beide Operanden von `cp` Dateien sind und beide Dateien existieren, überschreibt `cp` die zweite Datei mit einer Kopie der ersten Datei. Wir wollen das üben, indem wir die Datei `issue` mit der Datei `hostname` überschreiben:

```
$ cd ~/linux_essentials-2.4/copying/files
$ ls
hostname  issue  nsswitch.conf
$ cat hostname
mycomputer
$ cat issue
Debian GNU/Linux 9 \n \l

$ cp hostname issue
$ cat issue
mycomputer
```

Versuchen wir nun, eine Kopie des Verzeichnisses `files` im Verzeichnis `directories` zu erstellen:

```
$ cd ~/linux_essentials-2.4/copying
$ cp files directories
cp: omitting directory ‘files’
```

Wie Sie sehen, funktioniert `cp` standardmäßig nur bei einzelnen Dateien. Um ein Verzeichnis zu kopieren, verwenden Sie die Option `-r`. Beachten Sie, dass die Option `-r` bewirkt, dass `cp` den Inhalt des Verzeichnisses, das Sie kopieren, ebenfalls kopiert:

```
$ cp -r files directories
$ find
.
```

```
./files
./files/nsswitch.conf
./files/fstab
./files/hostname
./directories
./directories/files
./directories/files/nsswitch.conf
./directories/files/fstab
./directories/files/hostname
```

Sehen Sie, dass, wenn ein bestehendes Verzeichnis als Ziel verwendet wurde, `cp` eine Kopie des Quellverzeichnisses innerhalb des Verzeichnisses erstellt? Wenn das Ziel nicht existiert, wird es erstellt und mit dem Inhalt des Quellverzeichnisses gefüllt:

```
$ cp -r files files2
$ find
.
./files
./files/nsswitch.conf
./files/fstab
./files/hostname
./directories
./directories/files
./directories/files/nsswitch.conf
./directories/files/fstab
./directories/files/hostname
./files2
./files2/nsswitch.conf
./files2/fstab
./files2/hostname
```

Globbing

Was allgemein als *Globbing* bezeichnet wird, ist eine einfache Musterabgleichssprache. Kommandozeilen-Shells auf Linux-Systemen nutzen diese Sprache, um auf Gruppen von Dateien zu verweisen, deren Namen einem bestimmten Muster entsprechen. POSIX.1-2017 spezifiziert die folgenden Musterabgleichszeichen:

`*`

Entspricht einer beliebigen Anzahl von Zeichen, einschließlich kein Zeichen

`?`

Entspricht genau einem beliebigen Zeichen

`[]`

Entspricht einer Klasse von Zeichen

Übersetzt bedeutet das, dass Sie Ihrer Shell vorgeben, ein Muster statt einer genauen Zeichenkette anzunehmen. Normalerweise geben Linux-Benutzer mehrere Dateien mit einem Glob an, anstatt jeden einzelnen Dateinamen einzugeben. Führen Sie die folgenden Befehle aus:

```
$ cd ~/linux_essentials-2.4/globs
$ ls
question1   question14  question2012  star10    star2002
question13  question15  question23    star1100  star2013
$ ls star1*
star10  star1100
$ ls star*
star10  star1100  star2002  star2013
$ ls star2*
star2002  star2013
$ ls star2*2
star2002
$ ls star2013*
star2013
```

Die Shell erweitert `*` zu einer beliebigen Anzahl von Zeichen, so dass `star*` in diesem Kontext alles bedeutet, was mit `star` beginnt. Wenn Sie den Befehl `ls star*` ausführen, führt Ihre Shell das Programm `ls` nicht mit dem Argument `star*` aus — sie sucht vielmehr nach Dateien im aktuellen Verzeichnis, die dem Muster `star*` (einschließlich nur `star`) entsprechen, und verwandelt jede Datei, die dem Muster entspricht, in ein Argument zu `ls`:

```
$ ls star*
```

ist in Bezug auf `ls` dasselbe wie

```
$ ls star10  star1100  star2002  star2013
```

Das Zeichen `*` bedeutet nichts für `ls`. Um dies zu beweisen, führen Sie den folgenden Befehl aus:

```
ls star\*
```

```
ls: cannot access star*: No such file or directory
```

Wenn Sie einem Zeichen ein `\` voranstellen, weisen Sie Ihre Shell an, es nicht zu interpretieren. In diesem Fall möchten Sie, dass `ls` das Argument `star*` hat anstatt den Glob `star*` zu erweitern.

Das `?` erweitert zu einem einzelnen beliebigen Zeichen. Versuchen Sie es selbst mit den folgenden Befehlen:

```
$ ls
question1   question14  question2012  star10    star2002
question13  question15  question23    star1100  star2013
$ ls question?
question1
$ ls question1?
question13  question14  question15
$ ls question?3
question13  question23
$ ls question13?
ls: cannot access question13?: No such file or directory
```

Die `[]` Klammern dienen dazu, Bereiche oder Klassen von Zeichen anzugeben. Die `[]` Klammern funktionieren wie in POSIX regulären Ausdrücken, mit der Ausahme, dass bei Globs das `^` anstelle von `!` verwendet wird.

Erstellen Sie einige Dateien, mit denen Sie experimentieren können:

```
$ mkdir brackets
$ cd brackets
$ touch file1 file2 file3 file4 filea fileb filec file5 file6 file7
```

Bereiche innerhalb von `[]` Klammern werden mit einem `-` ausgedrückt:

```
$ ls
file1  file2  file3  file4  file5  file6  file7  filea  fileb  filec
$ ls file[1-2]
file1  file2
$ ls file[1-3]
file1  file2  file3
```

Es können mehrere Bereiche angegeben werden:

```
$ ls file[1-25-7]
file1  file2  file5  file6  file7
$ ls file[1-35-6a-c]
file1  file2  file3  file5  file6  filea  fileb  filec
```

Eckige Klammern können auch verwendet werden, um einen bestimmten Satz von Zeichen anzugeben:

```
$ ls file[1a5]
file1  file5  filea
```

Sie können `^` als erstes Zeichen verwenden, um alles *außer* den folgenden Zeichen anzugeben:

```
$ ls file[^a]
file1  file2  file3  file4  file5  file6  file7  fileb  filec
```

Abschließend behandeln wir in dieser Lektion *Zeichenklassen*. Um eine Zeichenklasse anzugeben, verwenden Sie `[:Zeichenklasse:]`. Für die Klasse `digit`, die alle Ziffern umfasst, schreiben Sie beispielweise:

```
$ ls file[[:digit:]]
file1  file2  file3  file4  file5  file6  file7
$ touch file1a file11
$ ls file[[:digit:]a]
file1  file2  file3  file4  file5  file6  file7  filea
$ ls file[[:digit:]]a
file1a
```

Der Glob `file[[:digit:]a]` entspricht `file`, gefolgt von einer Ziffer oder `a`.

Die unterstützten Zeichenklassen hängen von Ihrer aktuellen Locale-Konfiguration ab. POSIX benötigt für alle Locales die folgenden Zeichenklassen:

`[:alnum:]`
: Buchstaben und Zahlen.

`[:alpha:]`
: Groß- oder Kleinbuchstaben.

`[:blank:]`
Leerzeichen und Tabs.

`[:cntrl:]`
Steuerzeichen, z.B. Backspace, Glocke, NAK, Escape.

`[:digit:]`
Zahlen (`0123456789`).

`[:graph:]`
Alle graphischen Zeichen (alle Zeichen außer `ctrl` und Leerzeichen)

`[:lower:]`
Kleinbuchstaben (`a-z`).

`[:print:]`
Druckbare Zeichen (`alnum`, `punct` und das Leerzeichen).

`[:punct:]`
Interpunktionszeichen, d.h. `!`, &, `"`.

`[:space:]`
Whitespace-Zeichen, z.B. Tabs, Leerzeichen, Zeilenumbrüche.

`[:upper:]`
Großbuchstaben (`A-Z`).

`[:xdigit:]`
Hexadezimale Zahlen (normalerweise `0123456789abcdefABCDEF`).

Geführte Übungen

1. Gegeben sei die folgende Umgebung. Markieren Sie die Verzeichnisse, die der Befehl `mkdir -p /tmp/outfiles/text/today /tmp/infiles/text/today` erzeugen würde.

```
$ pwd
/tmp
$ find
.
./outfiles
./outfiles/text
```

`/tmp`	
`/tmp/outfiles`	
`/tmp/outfiles/text`	
`/tmp/outfiles/text/today`	
`/tmp/infiles`	
`/tmp/infiles/text`	
`/tmp/infiles/text/today`	

2. Was bewirkt `-v` bei `mkdir`, `rm` und `cp`?

3. Was passiert, wenn Sie versehentlich versuchen, drei Dateien auf der gleichen Befehlszeile in eine bereits vorhandene Datei zu kopieren anstatt in ein Verzeichnis?

4. Was passiert, wenn Sie mit `mv` ein Verzeichnis in sich selbst verschieben?

5. Wie würden Sie alle Dateien in Ihrem aktuellen Verzeichnis löschen, die mit `old` beginnen?

6. Welche der folgenden Dateien würden mit `log_[a-z]_201?_*_01.txt` übereinstimmen?

`log_3_2017_Jan_01.txt`	
`log_+_2017_Feb_01.txt`	

`log_b_2007_Mar_01.txt`	
`log_f_201A_Wednesday_01.txt`	

7. Erstellen Sie ein paar Globs, die der folgenden Liste von Dateinamen entsprechen:

```
doc100
doc200
doc301
doc401
```

Offene Übungen

1. Verwenden Sie die Man Page `cp`, um herauszufinden, wie man eine Kopie einer Datei erstellt und die Berechtigungen und Änderungszeiten mit dem Original übereinstimmen.

2. Was bewirkt der Befehl `rmdir -p`? Experimentieren Sie damit und erklären Sie, wie er sich von `rm -r` unterscheidet.

3. FÜHREN SIE DIESEN BEFEHL NICHT AUS: Was, denken Sie, bewirkt `rm -ri /*`? (EHRLICH, VERSUCHEN SIE NICHT, DAS ZU TUN!)

4. Ist es möglich, außer mit `-i` zu verhindern, dass `mv` Zieldateien überschreibt?

5. Erklären Sie den Befehl `cp -u`.

Zusammenfassung

Die Linux-Befehlszeilenumgebung bietet Werkzeuge zur Verwaltung von Dateien, darunter `cp`, `mv`, `mkdir`, `rm`, `rm` und `rmdir`. Diese Werkzeuge, kombiniert mit Globs, ermöglichen es, sehr schnell viel Arbeit zu erledigen.

Viele Befehle haben die Option `-i`, die Sie vor der Ausführung, den Befehl zu bestätigen. Das sog. *Prompten* kann Ihnen viel Ärger ersparen, wenn Sie sich vertippt haben.

Viele Befehle haben die Option `-r`. In der Mathematik und Informatik ist eine rekursive Funktion eine Funktion, die sich selbst in ihrer Definition verwendet. Wenn es um Kommandozeilen-Tools geht, bedeutet sie in der Regel, den Befehl auf ein Verzeichnis und alles darin anzuwenden.

Befehle, die in dieser Lektion verwendet wurden:

`cat`
: Liest und gibt den Inhalt einer Datei aus.

`cp`
: Kopiert Dateien oder Verzeichnisse.

`echo`
: Gibt eine Zeichenkette aus.

`find`
: Geht durch einen Dateisystembaum und sucht nach Dateien, die einem bestimmten Satz von Kriterien entsprechen.

`ls`
: Zeigt Eigenschaften von Dateien und Verzeichnissen und listet den Inhalt eines Verzeichnisses auf.

`mkdir`
: Erstellt neue Verzeichnisse.

`mv`
: Verschiebt Dateien oder Verzeichnisse und benennt sie um.

`pwd`
: Gibt das aktuelle Arbeitsverzeichnis aus.

`rm`

Löscht Dateien oder Verzeichnisse.

`rmdir`

Löscht Verzeichnisse.

`touch`

Erstellt neue leere Dateien oder aktualisiert den Änderungszeitstempel einer bestehenden Datei.

Lösungen zu den geführten Übungen

1. Gegeben sei die folgende Umgebung. Markieren Sie die Verzeichnisse, die der Befehl `mkdir -p /tmp/outfiles/text/today /tmp/infiles/text/today` erzeugen würde.

```
$ pwd
/tmp
$ find
.
./outfiles
./outfiles/text
```

   Die markierten Verzeichnisse werden erstellt. Die Verzeichnisse `/tmp`, `/tmp/outfiles` und `/tmp/outfiles/text` sind bereits vorhanden, so dass `mkdir` sie ignorieren wird.

`/tmp`	
`/tmp/outfiles`	
`/tmp/outfiles/text`	
`/tmp/outfiles/text/today`	X
`/tmp/infiles`	X
`/tmp/infiles/text`	X
`/tmp/infiles/text/today`	X

2. Was bewirkt `-v` bei `mkdir`, `rm` und `cp`?

   Typischerweise schaltet `-v` die ausführliche Ausgabe ein. Es bewirkt, dass die jeweiligen Programme ausgeben, was sie tun, während sie es tun:

```
$ rm -v a b
removed 'a'
removed 'b'
$ mv -v a b
'a' -> 'b'
$ cp -v b c
'b' -> 'c'
```

3. Was passiert, wenn Sie versehentlich versuchen, drei Dateien auf der gleichen Befehlszeile in eine bereits vorhandene Datei zu kopieren anstatt in ein Verzeichnis?

`cp` weigert sich, etwas zu tun, und gibt eine Fehlermeldung aus:

```
$ touch a b c d
$ cp a b c d
cp: target 'd' is not a directory
```

4. Was passiert, wenn Sie mit `mv` ein Verzeichnis in sich selbst verschieben?

   Sie erhalten eine Fehlermeldung, die Ihnen mitteilt, dass `mv` das nicht tun kann.

```
$ mv a a
mv: cannot move 'a' to a subdirectory of itself, 'a/a'
```

5. Wie würden Sie alle Dateien in Ihrem aktuellen Verzeichnis löschen, die mit `old` beginnen?

   Sie würden den Glob `alt*` mit `rm` verwenden:

```
$ rm old*
```

6. Welche der folgenden Dateien würde mit `log_[a-z]_201?_*_01.txt` übereinstimmen?

`log_3_2017_Jan_01.txt`	
`log_+_2017_Feb_01.txt`	
`log_b_2007_Mar_01.txt`	
`log_f_201A_Wednesday_01.txt`	X

```
$ ls log_[a-z]_201?_*_01.txt
log_f_201A_Wednesday_01.txt
```

   `log_[a-z]` entspricht `log_`, gefolgt von jedem Kleinbuchstaben, so dass sowohl `log_f_201A_Wednesday_01.txt` als auch `log_b_2007_Mar_01.txt` übereinstimmen. `_201?` passt auf jedes einzelne Zeichen, daher passt nur `log_f_201A_Wednesday_01.txt`. Schließlich passt `*_01.txt` zu allem, was mit `_01.txt` endet, also passt unsere verbleibende Option.

7. Erstellen Sie ein paar Globs, die der folgenden Liste von Dateinamen entsprechen:

```
doc100
doc200
```

```
doc301
doc401
```

Es gibt mehrere Lösungen. Hier sind einige von ihnen:

```
doc*
doc[1-4]*
doc?0?
doc[1-4]0?
```

Lösungen zu den offenen Übungen

1. Verwenden Sie die Man Page `cp`, um herauszufinden, wie man eine Kopie einer Datei erstellt und die Berechtigungen und Änderungszeiten mit dem Original übereinstimmen.

   Sie würden die Option `-p` verwenden. Aus der Man Page:

   ```
   $ man cp
   -p     same as --preserve=mode,ownership,timestamps
   --preserve[=ATTR_LIST]
                preserve the specified attributes (default: mode,ownership,time-
                stamps), if  possible  additional  attributes:  context,  links,
                xattr, all
   ```

2. Was bewirkt der Befehl `rmdir -p`? Experimentieren Sie damit und erklären Sie, wie er sich von `rm -r` unterscheidet.

   Er bewirkt, dass sich `rmdir` ähnlich wie `mkdir -p` verhält. Wenn Sie einen Baum mit leeren Verzeichnissen übergeben, werden alle entfernt.

   ```
   $ find
   .
   ./a
   ./a/b
   ./a/b/c
   $ rmdir -p a/b/c
   $ ls
   ```

3. FÜHREN SIE DIESEN BEFEHL NICHT AUS: Was, denken Sie, bewirkt `rm -ri /*`? (EHRLICH, VERSUCHEN SIE NICHT, DAS ZU TUN!)

   Es werden alle Dateien und Verzeichnisse entfernt, die von Ihrem Benutzerkonto beschreibbar sind. Dazu gehören auch alle Netzwerk-Dateisysteme.

4. Ist es möglich, außer mit `-i` zu verhindern, dass `mv` Zieldateien überschreibt?

   Ja, die Option `-n` oder `--no-clobber` verhindert, dass `mv` Dateien überschreibt.

   ```
   $ cat a
   a
   $ cat b
   ```

```
b
$ mv -n a b
$ cat b
b
```

5. Erklären Sie den Befehl `cp -u`.

   Die Option `-u` bewirkt, dass `cp` eine Datei nur dann kopiert, wenn das Ziel fehlt oder älter als die Quelldatei ist.

```
$ ls -l
total 24K
drwxr-xr-x 123 emma student  12K Feb  2 05:34 ..
drwxr-xr-x   2 emma student 4.0K Feb  2 06:56 .
-rw-r--r--   1 emma student    2 Feb  2 06:56 a
-rw-r--r--   1 emma student    2 Feb  2 07:00 b
$ cat a
a
$ cat b
b
$ cp -u a b
$ cat b
b
$ cp -u a c
$ ls -l
total 12
-rw-r--r-- 1 emma student 2 Feb  2 06:56 a
-rw-r--r-- 1 emma student 2 Feb  2 07:00 b
-rw-r--r-- 1 emma student 2 Feb  2 07:00 c
```

Linux
Professional
Institute

Thema 3: Die Macht der Befehlszeile

3.1 Dateien mithilfe der Befehlszeile archivieren

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 3.1

Gewichtung

2

Hauptwissensgebiete

- Dateien, Verzeichnisse
- Archive, Komprimierung

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- `tar`
- verbreitete `tar`-Optionen
- `gzip`, `bzip2`, `xz`
- `zip`, `unzip`

3.1 Lesson 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	3 Die Macht der Befehlszeile
Lernziel:	3.1 Dateien mithilfe der Befehlszeile archivieren
Lektion:	1 von 1

Einführung

Komprimierung wird eingesetzt, um den Platzbedarf für einen bestimmten Datensatz zu reduzieren. Üblicherweise dient Komprimierung der Einsparung von Speicherplatz für eine Datei oder der Menge der über eine Netzwerkverbindung gesendeten Daten.

Kompression funktioniert durch das Ersetzen sich wiederholender Muster durch Daten. Angenommen in einem Roman kommen einige Wörter aus mehreren Zeichen extrem häufig vor, etwa das Wort “das”. Sie könnten die Größe des Romans deutlich reduzieren, indem Sie diese mehrstelligen Wörter und Muster durch Einzelzeichen ersetzen, zum Beispiel das “das” durch einen griechischen Buchstaben, der sonst im Text nicht vorkommt.

Kompression gibt es in zwei Varianten: *verlustfrei* (*lossless*) und *verlustbehaftet* (*lossy*). Dinge, die mit einem verlustfreien Algorithmus komprimiert wurden, lassen sich wieder in ihre ursprüngliche Form überführen. Daten, die mit einem verlustbehafteten Algorithmus komprimiert wurden, können nicht wiederhergestellt werden. Verlustbehaftete Algorithmen werden oft für Bilder, Videos und Audiodateien verwendet, bei denen der Qualitätsverlust für den Menschen unmerklich, für den Kontext irrelevant oder der Verlust den eingesparten Platz oder

Netzwerkdurchsatz wert ist.

Archivierungswerkzeuge dienen dazu, Dateien und Verzeichnisse in einer einzigen Datei zusammenzufassen, wie etwa Backups, Source Code und Langzeitarchive.

Archive und Komprimierung gehen meist einher. Einige Archivierungswerkzeuge komprimieren sogar standardmäßig ihren Inhalt, andere komprimieren ihren Inhalt optional. Einige Archivierungswerkzeuge müssen mit eigenständigen Komprimierungswerkzeugen kombiniert werden, wenn Sie den Inhalt komprimieren möchten.

Das gebräuchlichste Werkzeug zur Archivierung von Dateien auf Linux-Systemen ist `tar`. Die meisten Linux-Distributionen werden mit der GNU-Version von `tar` ausgeliefert, darum behandeln wir es auch in dieser Lektion. `tar` verwaltet lediglich die Archivierung von Dateien, komprimiert diese aber nicht.

Es gibt viele Komprimierungswerkzeuge unter Linux, einige gängige verlustfreie sind `bzip2`, `gzip` und `xz`. Alle drei finden Sie auf den meisten Systemen, können aber auch auf ein altes oder reduziertes System stoßen, auf dem `xz` oder `bzip` nicht installiert sind. Mit hoher Wahrscheinlichkeit werden Sie aber auf Dateien treffen, die mit allen drei Werkzeugen komprimiert wurden. Alle drei verwenden unterschiedliche Algorithmen, so dass eine mit einem Tool komprimierte Datei nicht von einem anderen dekomprimiert werden kann. Bei allen Komprimierungswerkzeugen muss man Kompromisse eingehen: Wenn Sie eine hohe Kompressionsrate wünschen, dauert es länger, die Datei zu komprimieren und zu dekomprimieren. Das liegt daran, dass eine höhere Kompression mehr Aufwand erfordert, komplexere Muster zu finden. Die genannten Tools komprimieren Daten, können aber keine Archive mit mehreren Dateien erstellen.

Selbständige Komprimierungswerkzeuge sind in der Regel auf Windows-Systemen nicht verfügbar. Windows-Archivierungs- und Komprimierungswerkzeuge sind meist miteinader kombiniert. Bedenken Sie dies, wenn Sie Linux- und Windows-Systeme haben, die Dateien gemeinsam nutzen müssen.

Linux-Systeme haben auch Werkzeuge für die Verwaltung von `.zip`-Dateien, die auf Windows-Systemen üblich sind: Sie heißen `zip` und `unzip`, werden aber nicht auf allen Systemen standardmäßig installiert, so dass Sie sie gegebenenfalls installieren müssen. Glücklicherweise finden sie sich aber in den Paketsammlungen der meisten Distributionen.

Komprimierungswerkzeuge

Wie viel Speicherplatz durch die Komprimierung von Dateien eingespart wird, hängt von einigen Faktoren ab: der Art der zu komprimierenden Daten, dem Algorithmus zur Komprimierung der

Daten und der Komprimierungsstufe. Nicht alle Algorithmen unterstützen unterschiedliche Komprimierungsstufen.

Legen wir zunächst einige Testdateien zur Komprimierung an:

```
$ mkdir ~/linux_essentials-3.1
$ cd ~/linux_essentials-3.1
$ mkdir compression archiving
$ cd compression
$ cat /etc/* > bigfile 2> /dev/null
```

Jetzt erstellen wir drei Kopien dieser Datei:

```
$ cp bigfile bigfile2
$ cp bigfile bigfile3
$ cp bigfile bigfile4
$ ls -lh
total 2.8M
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile2
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile3
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile4
```

Nun komprimieren wir die Dateien mit jedem der oben genannten Kompressionstools:

```
$ bzip2 bigfile2
$ gzip bigfile3
$ xz bigfile4
$ ls -lh
total 1.2M
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile
-rw-r--r-- 1 emma emma 170K Jun 23 08:08 bigfile2.bz2
-rw-r--r-- 1 emma emma 179K Jun 23 08:08 bigfile3.gz
-rw-r--r-- 1 emma emma 144K Jun 23 08:08 bigfile4.xz
```

Vergleichen Sie die Größen der komprimierten Dateien mit der unkomprimierten Datei namens `bigfile` und beachten Sie, dass die Komprimierungswerkzeuge Erweiterungen zu den Dateinamen hinzugefügt und die unkomprimierten Dateien entfernt haben.

Verwenden Sie `bunzip2`, `gunzip` oder `unxz`, um die Dateien zu dekomprimieren:

```
$ bunzip2 bigfile2.bz2
$ gunzip bigfile3.gz
$ unxz bigfile4.xz
$ ls -lh
total 2.8M
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile2
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile3
-rw-r--r-- 1 emma emma 712K Jun 23 08:20 bigfile4
```

Beachten Sie, dass nun die komprimierte Datei gelöscht wird, sobald sie dekomprimiert wurde.

Einige Komprimierungswerkzeuge unterstützen unterschiedliche Komprimierungsstufen. Eine höhere Komprimierungsstufe erfordert in der Regel mehr Speicher und CPU-Zyklen, führt aber zu einer kleineren komprimierten Datei. Für niedrigere Stufen gilt entsprechend das Gegenteil. Hier ein Beispiel mit `xz` und `gzip`:

```
$ cp bigfile bigfile-gz1
$ cp bigfile bigfile-gz9
$ gzip -1 bigfile-gz1
$ gzip -9 bigfile-gz9
$ cp bigfile bigfile-xz1
$ cp bigfile bigfile-xz9
$ xz -1 bigfile bigfile-xz1
$ xz -9 bigfile bigfile-xz9
$ ls -lh bigfile bigfile-* *
total 3.5M
-rw-r--r-- 1 emma emma 712K Jun 23 08:08 bigfile
-rw-r--r-- 1 emma emma 205K Jun 23 13:14 bigfile-gz1.gz
-rw-r--r-- 1 emma emma 178K Jun 23 13:14 bigfile-gz9.gz
-rw-r--r-- 1 emma emma 156K Jun 23 08:08 bigfile-xz1.xz
-rw-r--r-- 1 emma emma 143K Jun 23 08:08 bigfile-xz9.xz
```

Es ist nicht notwendig, eine Datei bei jeder Verwendung zu dekomprimieren. Komprimierungswerkzeuge verfügen in der Regel über spezielle Versionen gängiger Werkzeuge zum Lesen von Textdateien, wie z.B. `gzip` mit `cat`, `grep`, `diff`, `less`, `more` und einigen anderen. Bei `gzip` ist den Werkzeugen ein `z` vorangestellt, bei `bzip2` das `bz` und bei `xz` das Präfix `xz`:

```
$ cp /etc/hosts ./
$ gzip hosts
$ zcat hosts.gz
```

```
127.0.0.1   localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Archivierungswerkzeuge

Das Programm `tar` ist wohl das am weitesten verbreitete Archivierungswerkzeug auf Linux-Systemen. Der Name ist übrigens eine Abkürzung für "Tape Archive". Dateien, die mit `tar` erstellt wurden, werden oft als *tar balls* bezeichnet. Es ist sehr verbreitet, den Quellcode von Anwendungen in tar balls zur Verfügung zu stellen.

Die GNU-Version von `tar`, mit der Linux-Distributionen ausgeliefert werden, hat viele Optionen. In dieser Lektion stellen wir die am häufigsten verwendeten vor.

Beginnen wir mit der Erstellung eines Archivs der zur Komprimierung vorgesehenen Dateien:

```
$ cd ~/linux_essentials-3.1
$ tar cf archiving/3.1.tar compression
```

Die Option `c` weist `tar` an, eine neue Archivdatei zu erstellen, und die Option `f` ist der Name der zu erstellenden Datei: Das Argument unmittelbar nach den Optionen ist immer der Name der zu bearbeitenden Datei. Die übrigen Argumente sind die Pfade zu allen Dateien oder Verzeichnissen, die Sie der Datei hinzufügen, auflisten oder extrahieren möchten. In dem Beispiel fügen wir das Verzeichnis `compression` samt Inhalt dem Archiv hinzu.

Um den Inhalt eines tar balls zu sehen, verwenden Sie die Option `t` von `tar`:

```
$ tar -tf 3.1.tar
compression/
compression/bigfile-xz1.xz
compression/bigfile-gz9.gz
compression/hosts.gz
compression/bigfile2
compression/bigfile
compression/bigfile-gz1.gz
compression/bigfile-xz9.xz
compression/bigfile3
```

```
compression/bigfile4
```

Beachten Sie, wie den Optionen `-` vorangestellt wird. Im Gegensatz zu den meisten Programmen ist bei `tar` das `-` bei der Angabe von Optionen nicht erforderlich, hat aber auch keine negativen Folgen.

NOTE Nutzen Sie die Option `-v`, um `tar` die Namen der Dateien ausgeben zu lassen, mit denen es beim Erstellen oder Extrahieren eines Archivs arbeitet.

Lassen Sie uns nun die Datei entpacken:

```
$ cd ~/linux_essentials-3.1/archiving
$ ls
3.1.tar
$ tar xf 3.1.tar
$ ls
3.1.tar  compression
```

Angenommen, Sie benötigen nur eine Datei aus dem Archiv — in diesem Fall geben Sie diese hinter dem Dateinamen des Archivs an. Bei Bedarf können Sie auch mehrere Dateien angeben:

```
$ cd ~/linux_essentials-3.1/archiving
$ rm -rf compression
$ ls
3.1.tar
$ tar xvf 3.1.tar compression/hosts.gz
compression/
compression/bigfile-xz1.xz
compression/bigfile-gz9.gz
compression/hosts.gz
compression/bigfile2
compression/bigfile
compression/bigfile-gz1.gz
compression/bigfile-xz9.xz
compression/bigfile3
compression/bigfile4
$ ls
3.1.tar  compression
$ ls compression
hosts.gz
```

Mit Ausnahme von absoluten Pfaden (also solchen, die mit `/` beginnen), behalten `tar`-Dateien den gesamten Pfad zu Dateien bei, wenn sie erstellt werden. Da die Datei `3.1.tar` mit einem einzigen Verzeichnis erstellt wurde, wird dieses Verzeichnis beim Extrahieren relativ zu Ihrem aktuellen Arbeitsverzeichnis erstellt. Ein weiteres Beispiel soll dies verdeutlichen:

```
$ cd ~/linux_essentials-3.1/archiving
$ rm -rf compression
$ cd ../compression
$ tar cf ../tar/3.1-nodir.tar *
$ cd ../archiving
$ mkdir untar
$ cd untar
$ tar -xf ../3.1-nodir.tar
$ ls
bigfile   bigfile3  bigfile-gz1.gz  bigfile-xz1.xz  hosts.gz
bigfile2  bigfile4  bigfile-gz9.gz  bigfile-xz9.xz
```

TIP Wenn Sie den absoluten Pfad in einer `tar` Datei verwenden möchten, nutzen Sie die Option `P`. Beachten Sie, dass dies wichtige Dateien überschreiben und Fehler auf Ihrem System verursachen kann.

`tar` kann auch Komprimierung und Dekomprimierung von Archiven on the fly durchführen, indem es eines der oben beschriebenen Komprimierungswerkzeuge aufruft. Dazu ist lediglich die dem Komprimierungsalgorithmus entsprechende Option hinzuzufügen. Die am häufigsten verwendeten sind `j`, `J` und `z` für `bzip2`, `xz` und `gzip`. Im Folgenden einige Beispiele:

```
$ cd ~/linux_essentials-3.1/compression
$ ls
bigfile   bigfile3  bigfile-gz1.gz  bigfile-xz1.xz  hosts.gz
bigfile2  bigfile4  bigfile-gz9.gz  bigfile-xz9.xz
$ tar -czf gzip.tar.gz bigfile bigfile2 bigfile3
$ tar -cjf bzip2.tar.bz2 bigfile bigfile2 bigfile3
$ tar -cJf xz.tar.xz bigfile bigfile2 bigfile3
$ ls -l | grep tar
-rw-r--r-- 1 emma emma  450202 Jun 27 05:56 bzip2.tar.bz2
-rw-r--r-- 1 emma emma  548656 Jun 27 05:55 gzip.tar.gz
-rw-r--r-- 1 emma emma  147068 Jun 27 05:56 xz.tar.xz
```

Sie sehen, dass im Beispiel die `.tar`-Dateien unterschiedlich groß sind. Das zeigt, dass sie erfolgreich komprimiert wurden. Wenn Sie komprimierte `.tar`-Archive erstellen, sollten Sie immer eine zweite Dateierweiterung hinzufügen, die den von Ihnen verwendeten Algorithmus

angibt: `.xz`, `.bz` und `.gz` für `xz`, `bzip2` bzw. `gzip`. Manchmal werden verkürzte Erweiterungen wie `.tgz` verwendet.

Es ist möglich, Dateien zu bereits vorhandenen unkomprimierten `tar`-Archiven hinzuzufügen, und zwar über die Option `u`. Wenn Sie versuchen, sie zu einem komprimierten Archiv hinzuzufügen, erhalten Sie eine Fehlermeldung.

```
$ cd ~/linux_essentials-3.1/compression
$ ls
bigfile   bigfile3  bigfile-gz1.gz  bigfile-xz1.xz  bzip2.tar.bz2  hosts.gz
bigfile2  bigfile4  bigfile-gz9.gz  bigfile-xz9.xz  gzip.tar.gz    xz.tar.xz
$ tar cf plain.tar bigfile bigfile2 bigfile3
$ tar tf plain.tar
bigfile
bigfile2
bigfile3
$ tar uf plain.tar bigfile4
$ tar tf plain.tar
bigfile
bigfile2
bigfile3
bigfile4
$ tar uzf gzip.tar.gz bigfile4
tar: Cannot update compressed archives
Try 'tar --help' or 'tar --usage' for more information.
```

Verwalten von ZIP-Dateien

Windows-Maschinen verfügen oft nicht über Anwendungen zur Verarbeitung von tar balls und andere auf Linux-Systemen übliche Kompressionstools. Wenn Sie mit Windows-Systemen interagieren müssen, können Sie ZIP-Dateien verwenden. Eine ZIP-Datei ist eine Archivdatei, die einer komprimierten `tar`-Datei ähnelt.

Sie können die Programme `zip` und `unzip` für die Arbeit mit ZIP-Dateien auf Linux-Systemen nutzen. Das folgende Beispiel zeigt, was Sie dafür benötigen:

```
$ cd ~/linux_essentials-3.1
$ mkdir zip
$ cd zip/
$ mkdir dir
$ touch dir/file1 dir/file2
```

Jetzt verwenden wir `zip`, um diese Dateien in eine ZIP-Datei zu packen:

```
$ zip -r zipfile.zip dir
  adding: dir/ (stored 0%)
  adding: dir/file1 (stored 0%)
  adding: dir/file2 (stored 0%)
$ rm -rf dir
```

Anschließend entpacken wir die ZIP-Datei wieder:

```
$ ls
zipfile.zip
$ unzip zipfile.zip
Archive:  zipfile.zip
   creating: dir/
 extracting: dir/file1
 extracting: dir/file2
$ find
.
./zipfile.zip
./dir
./dir/file1
./dir/file2
```

Wenn Sie Verzeichnisse zu ZIP-Dateien hinzufügen, bewirkt die Option `-r`, dass `zip` auch den Inhalt eines Verzeichnisses enthält. Ohne die Option bliebe das Verzeichnis in der ZIP-Datei leer.

Geführte Übungen

1. Welches der folgenden Tools wurde entsprechend der jeweiligen Erweiterung zur Erstellung dieser Dateien verwendet?

Dateiname	`tar`	`gzip`	`bzip2`	`xz`
`archive.tar`				
`archive.tgz`				
`archive.tar.xz`				

2. Welche dieser Dateien sind entsprechend der jeweiligen Erweiterung Archive und welche sind komprimiert?

Dateiname	**Archiv**	**Komprimiert**
`file.tar`		
`file.tar.bz2`		
`file.zip`		
`file.xz`		

3. Wie würden Sie eine Datei zu einer mit `gzip` komprimierten `tar`-Datei hinzufügen?

4. Welche `tar`-Option weist `tar` an, den führenden `/` in absolute Pfade aufzunehmen?

5. Unterstützt `zip` verschiedene Kompressionsstufen?

Offene Übungen

1. Unterstützt `tar` beim Extrahieren von Dateien Globs in der Dateiliste?

2. Wie können Sie sicherstellen, dass eine dekomprimierte Datei mit der Datei identisch ist, bevor sie komprimiert wurde?

3. Was passiert, wenn Sie versuchen, eine Datei aus einem `tar`-Archiv zu extrahieren, die bereits auf Ihrem Dateisystem existiert?

4. Wie würden Sie die Datei `archive.tgz` extrahieren, ohne die `tar`-Option `z` zu verwenden?

Zusammenfassung

Linux-Systeme verfügen über mehrere Komprimierungs- und Archivierungswerkzeuge. In dieser Lektion wurden die gängigsten behandelt. Das meistgenutzte Archivierungswerkzeug ist `tar`. Wenn eine Interaktion mit Windows-Systemen erforderlich ist, können `zip` und `unzip` ZIP-Dateien erstellen und extrahieren.

Der Befehl `tar` hat einige Optionen, die man sich merken sollte: `x` steht für das Extrahieren, `c` für das Erstellen, `t` für das Anzeigen der Inhalte und `u` für das Hinzufügen oder Ersetzen von Dateien. `v` listet die Dateien auf, die von `tar` beim Erstellen oder Entpacken eines Archivs verarbeitet werden.

Das Repository der typischen Linux-Distribution verfügt über viele Komprimierungswerkzeuge, darunter `gzip`, `bzip2` und `xz`. Komprimierungsalgorithmen unterstützen oft verschiedene Ebenen, mit denen Sie die Geschwindigkeit oder Dateigröße optimieren. Dateien lassen sich mit `gunzip`, `bunzip2` und `unxz` dekomprimieren.

Komprimierungswerkzeuge umfassen häufig Programme, die sich wie gängige Textdateiwerkzeuge verhalten, mit dem Unterschied, dass sie mit komprimierten Dateien arbeiten. Einige von ihnen sind `zcat`, `bzcat` und `xzcat`. Komprimierungswerkzeuge werden normalerweise mit Programmen mit der Funktionalität von `grep`, `more`, `less`, `diff` und `cmp` ausgeliefert.

Befehle, die in den Übungen verwendet werden:

`bunzip2`
: Dekomprimiert eine mit `bzip2` komprimierte Datei.

`bzcat`
: Gibt den Inhalt einer mit `bzip` komprimierten Datei aus.

`bzip2`
: Komprimiert Dateien mit dem `bzip2`-Algorithmus und -Format.

`gunzip`
: Dekomprimiert eine mit `gzip` komprimierte Datei.

`gzip`
: Komprimiert Dateien mit dem `gzip`-Algorithmus und -Format.

`tar`

Erstellt, aktualisiert, listet und extrahiert `tar`-Archive.

`unxz`

Dekomprimiert eine mit `xz` komprimierte Datei.

`unzip`

Dekomprimiert und extrahiert Inhalte aus einer ZIP-Datei.

`xz` Komprimiert Dateien mit dem `xz`-Algorithmus und -Format.

`zcat`

Gibt den Inhalt einer mit `gzip` komprimierten Datei aus.

`zip`

Erzeugt und komprimiert ZIP-Archive.

Lösungen zu den geführten Übungen

1. Welches der folgenden Tools wurde entsprechend der jeweiligen Erweiterung zur Erstellung dieser Dateien verwendet?

Dateiname	`tar`	`gzip`	`bzip2`	`xz`
`archive.tar`	X			
`archive.tgz`	X	X		
`archive.tar.xz`	X			X

2. Welche dieser Dateien sind entsprechend der jeweiligen Erweiterung Archive und welche sind komprimiert?

Dateiname	**Archiv**	**Komprimiert**
`file.tar`	X	
`file.tar.bz2`	X	X
`file.zip`	X	X
`file.xz`		X

3. Wie würden Sie eine Datei zu einer mit `gzip` komprimierten `tar` Datei hinzufügen?

   Sie dekomprimieren die Datei mit `gunzip`, fügen die Datei mit `tar uf` hinzu und komprimieren sie dann mit `gzip`.

4. Welche `tar`-Option weist `tar` an, den führenden `/` in absolute Pfade aufzunehmen?

   Die Option `-P`. Aus der Man Page:

```
-P, --absolute-names
       Don't strip leading slashes from file names when creating  archives
```

5. Unterstützt `zip` verschiedene Kompressionsstufen?

   Ja. Nutzen Sie `-#` und ersetzen Sie `#` mit einer Zahl von 0-9. Aus der Man Page:

```
-#
(-0, -1, -2, -3, -4, -5, -6, -7, -8, -9)
      Regulate the speed of compression using the specified  digit  #,
```

```
where -0 indicates no compression (store all files), -1 indi-
cates the fastest compression speed (less compression) and -9
indicates the slowest compression speed (optimal compression,
ignores the suffix list). The default compression level is -6.

Though still being worked, the intention is this setting will
control compression speed for all compression methods. Cur-
rently only deflation is controlled.
```

Lösungen zu den offenen Übungen

1. Unterstützt `tar` beim Extrahieren von Dateien Globs in der Dateiliste?

   Ja, mit der Option `--wildcards`. `--wildcards` muss direkt nach der `tar`-Datei folgen, wenn Sie die Optionen ohne Bindestrich verwenden, z.B.:

   ```
   $ tar xf tarfile.tar --wildcards dir/file*
   $ tar --wildcards -xf tarfile.tar dir/file*
   ```

2. Wie können Sie sicherstellen, dass eine dekomprimierte Datei mit der Datei identisch ist, bevor sie komprimiert wurde?

   Sie müssen mit den in dieser Lektion behandelten Werkzeugen nichts tun. Alle drei enthalten Prüfsummen in ihrem Dateiformat, die beim Dekomprimieren überprüft werden.

3. Was passiert, wenn Sie versuchen, eine Datei aus einem `tar`-Archiv zu extrahieren, die bereits auf Ihrem Dateisystem existiert?

   Die Datei auf Ihrem Dateisystem wird mit der Version überschrieben, die sich in der `tar`-Datei befindet.

4. Wie würden Sie die Datei `archive.tgz` extrahieren, ohne die `tar`-Option `z` zu verwenden?

   Zunächst mit `gunzip` dekomprimieren:

   ```
   $ gunzip archive.tgz
   $ tar xf archive.tar
   ```

3.2 Daten in Dateien suchen und extrahieren

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 3.2

Gewichtung

3

Hauptwissensgebiete

- Command line pipes
- I/O redirection
- Basic Regular Expressions using `.`, `[ ]`, `*`, und `?`

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

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

3.2 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	3 Die Macht der Befehlszeile
Lernziel:	3.2 Daten in Dateien suchen und extrahieren
Lektion:	1 von 2

Einführung

In dieser Lektion konzentrieren wir uns auf die Umleitung oder Übergabe von Informationen von einer Quelle zur anderen mit Hilfe spezifischer Werkzeuge. Die Linux-Befehlszeile leitet Informationen über bestimmte Standardkanäle um. Als Standardeingabe (*Standard Input*, *stdin* oder Kanal 0) eines Befehls gilt die Tastatur, und als Standardausgabe (*Standard Output*, *stdout* oder Kanal 1) der Bildschirm. Es gibt einen weiteren Kanal, der dazu dient, die Fehlerausgabe (*Standard Error*, *stderr* oder Kanal 2) eines Befehls oder die Fehlermeldungen eines Programms umzuleiten. Der Ein- und/oder Ausgabe kann umgeleitet werden.

Wenn wir einen Befehl ausführen, wollen wir manchmal bestimmte Informationen an den Befehl senden oder die Ausgabe in eine bestimmte Datei umleiten. Diese Funktionalitäten behandeln die nächsten beiden Abschnitte.

I/O-Umleitung

Die I/O-Umleitung ermöglicht es dem Benutzer, Informationen von oder zu einem Befehl mithilfe einer Textdatei umzuleiten. Wie bereits beschrieben, kann die Standardeingabe, -ausgabe und -fehlerausgabe umgeleitet werden, und die Informationen können aus Textdateien übernommen

werden.

Umleitung der Standardausgabe

Um die Standardausgabe in eine Datei umzuleiten, nutzen wir den Operator `>`, gefolgt vom Namen der Datei. Wenn die Datei nicht existiert, wird eine neue erstellt, andernfalls überschreiben die Informationen die bestehende Datei.

Um den Inhalt der Datei, die wir gerade erstellt haben, zu sehen, können wir den Befehl `cat` verwenden, der standardmäßig den Inhalt einer Datei auf dem Bildschirm anzeigt. Auf der Man Page erfahren Sie mehr über dessen Funktionalitäten.

Das folgende Beispiel zeigt die Funktionalität des Operators. Zunächst wird eine neue Datei mit dem Text "`Hello World!`" erstellt.:

```
$ echo "Hello World!" > text
$ cat text
Hello World!
```

Beim zweiten Aufruf wird die gleiche Datei mit dem neuen Text überschrieben:

```
$ echo "Hello!" > text
$ cat text
Hello!
```

Um neue Informationen am Ende der Datei hinzuzufügen, nutzen wir den Operator `>>` , der auch eine neue Datei erstellt, wenn er keine existierende findet.

Das erste Beispiel zeigt das Hinzufügen des Textes. Wie man sieht, wurde der neue Text in der folgenden Zeile hinzugefügt:

```
$ echo "Hello to you too!" >> text
$ cat text
Hello!
Hello to you too!
```

Das zweite Beispiel zeigt, dass eine neue Datei erstellt wird:

```
$ echo "Hello to you too!" >> text2
$ cat text2
```

```
Hello to you too!
```

Umleitung der Standardfehlerausgabe (Standard Error)

Um nur die Fehlermeldungen umzuleiten, nutzt ein User den Operator `2>`, gefolgt vom Namen der Datei, in die die Fehler geschrieben werden sollen. Wenn die Datei nicht existiert, wird eine neue erstellt, andernfalls wird die Datei überschrieben.

Wie bereits erläutert, ist der Kanal für die Umleitung der Standardfehlerausgabe *Kanal 2*. Bei der Umleitung der Standardfehlerausgabe muss der Kanal angegeben werden, im Gegensatz zu den anderen Standardausgaben, bei denen *Kanal 1* standardmäßig gesetzt ist. Der folgende Befehl sucht beispielsweise nach einer Datei oder einem Verzeichnis namens games und schreibt nur den Fehler in die Datei `text-error`, während er die Standardausgabe auf dem Bildschirm anzeigt:

```
$ find /usr games 2> text-error
/usr
/usr/share
/usr/share/misc
---------Omitted output----------
/usr/lib/libmagic.so.1.0.0
/usr/lib/libdns.so.81
/usr/games
$ cat text-error
find: `games': No such file or directory
```

NOTE Weitere Informationen über den Befehl `find` finden Sie in der Man Page.

Der folgende Befehl wird ohne Fehler ausgeführt, daher werden keine Informationen in die Datei `text-error` geschrieben:

```
$ sort /etc/passwd 2> text-error
$ cat text-error
```

Wie die Standardausgabe kann auch die Standardfehlerausgabe an eine Datei mit dem Operator `2>>` angehängt werden. Der neue Fehler wird am Ende der Datei hinzugefügt. Wenn die Datei nicht existiert, wird eine neue erstellt. Das erste Beispiel zeigt das Hinzufügen der neuen Informationen in die Datei, während das zweite Beispiel zeigt, dass der Befehl eine neue Datei erstellt, falls keine Datei dieses Namens gefunden wird:

```
$ sort /etc 2>> text-error
$ cat text-error
sort: read failed: /etc: Is a directory
```

```
$ sort /etc/shadow 2>> text-error2
$ cat text-error2
sort: open failed: /etc/shadow: Permission denied
```

Bei dieser Art der Umleitung werden nur Fehlermeldungen in die Datei umgeleitet, die normale Ausgabe erscheint auf dem Bildschirm bzw. geht an die Standardausgabe (*stdout*).

Es gibt eine bestimmte Datei, die technisch gesehen ein "`Daten-Mülleimer`", auch *bit bucket* genannt, ist, das heißt sie akzeptiert Eingaben, macht aber nichts damit: `/dev/null`. Sie können alle unwichtigen Informationen, die Sie nicht anzeigen oder in einer Datei speichern möchten, wie im folgenden Beispiel umleiten:

```
$ sort /etc 2> /dev/null
```

Umleitung der Standardeingabe (Standard Input)

Diese Art der Umleitung dient dazu, einem Befehl Daten aus einer bestimmten Datei statt über die Tastatur zu übergeben. Das geschieht über den Operator `<`, wie im folgenden Beispiel zu sehen:

```
$ cat < text
Hello!
Hello to you too!
```

Die Umleitung der Standardeingabe wird normalerweise bei Befehlen verwendet, die keine Argumente akzeptieren. Der Befehl `tr` ist einer von diesen. Er dient dazu, Dateiinhalte zu übersetzen, indem man die Zeichen in einer Datei auf bestimmte Weise ändert, etwa durch das Löschen eines bestimmten Zeichens aus einer Datei. Das folgende Beispiel zeigt das Löschen des Zeichens `l`:

```
$ tr -d "l" < text
Heo!
Heo to you too!
```

Weitere Informationen finden Sie in der Man Page von `tr`.

Here Documents

Anders als bei den Ausgabeumleitungen verhält sich der Operator `<<`. Dieser Input Stream wird auch *here document* genannt. Ein *here document* repräsentiert einen Text- oder Code-Block, der an den Befehl oder das interaktive Programm übergeben werden kann. Verschiedene Skriptsprachen wie `bash`, `sh` und `csh` können so Eingaben direkt von der Kommandozeile übernehmen, ohne Textdateien zu nutzen.

Wie im folgenden Beispiel zu sehen, dient der Operator dazu, Daten an den Befehl zu übergeben, wobei das folgende Wort keinen Dateinamen bezeichnet. Das Wort wird vielmehr als Trennzeichen der Eingabe interpretiert und nicht als Inhalt berücksichtigt; daher wird es von `cat` nicht angezeigt:

```
$ cat << hello
> hey
> ola
> hello
hey
ola
```

Weitere Informationen finden Sie in der Man Page des Befehls `cat`.

Kombinationen

Die erste Kombination leitet Standardausgabe und Standardfehlerausgabe in eine einzige Datei um. Dazu nutzen wir die Operatoren `&>` und `&>>`, wobei `&` die Kombination aus *Kanal 1* und *Kanal 2* repräsentiert. Der erste Operator überschreibt den Inhalt einer vorhandenen Datei, der zweite fügt die neuen Informationen am Ende einer existierenden Datei hinzu. Beide Operatoren ermöglichen, wie in den vorangegangenen Abschnitten, die Erstellung der neuen Datei, wenn sie nicht bereits existiert:

```
$ find /usr admin &> newfile
$ cat newfile
/usr
/usr/share
/usr/share/misc
---------Omitted output----------
/usr/lib/libmagic.so.1.0.0
/usr/lib/libdns.so.81
/usr/games
find: `admin': No such file or directory
```

```
$ find /etc/calendar &>> newfile
$ cat newfile
/usr
/usr/share
/usr/share/misc
---------Omitted output----------
/usr/lib/libmagic.so.1.0.0
/usr/lib/libdns.so.81
/usr/games
find: `admin': No such file or directory
/etc/calendar
/etc/calendar/default
```

Schauen wir uns ein Beispiel mit dem Befehl `cut` an:

```
$ cut -f 3 -d "/" newfile
$ cat newfile

share
share
share
---------Omitted output----------
lib
games
find: `admin': No such file or directory
calendar
calendar
find: `admin': No such file or directory
```

Der Befehl `cut` schneidet mit der Option `-f` bestimmte Felder aus der Eingabedatei , in unserem Fall das dritte Feld. Damit der Befehl das Feld findet, ist zudem ein Trennzeichen mit der Option `-d` angegeben, in unserem Fall das Zeichen `/`.

Um mehr über den Befehl `cut` zu erfahren, konsultieren Sie die Man Page.

Pipes

Eine Umleitung wird meist verwendet, um das Ergebnis eines Befehls zu speichern und anschließend von einem anderen Befehl verarbeiten zu lassen. Diese Zwischenprozesse können sehr mühsam und kompliziert werden, wenn die Daten mehrere Prozesse durchlaufen. Um dies zu vermeiden, lassen sich Befehle direkt über Pipes verknüpfen. Die Ausgabe des ersten Befehls wird damit automatisch zur Eingabe des zweiten Befehls. Diese Verbindung wird über den

Operator `|` (senkrechter Strich) hergestellt:

```
$ cat /etc/passwd | less
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
:
```

Im obigen Beispiel ändert der Befehl `less` nach dem Pipe-Operator die Art und Weise, wie die Datei angezeigt wird. Der Befehl `less` zeigt die Textdatei so an, dass der Benutzer eine Zeile auf und ab scrollen kann. `less` wird standardmäßig auch verwendet, um die Man Pages anzuzeigen, wie in den vorherigen Lektionen beschrieben.

Es ist möglich, mehrere Pipes gleichzeitig zu verwenden. Die Zwischenbefehle, die Eingaben empfangen, verarbeiten und dann eine Ausgabe erzeugen, heißen *Filter*. Nehmen wir den Befehl `ls -l` und versuchen wir, die Anzahl der Wörter aus den ersten 10 Zeilen der Ausgabe zu zählen. Dazu nutzen wir den Befehl `head`, der standardmäßig die ersten 10 Zeilen einer Datei anzeigt, und dann den Befehl `wc`, der die Wörter zählt:

```
$ ls -l | head | wc -w
10
```

Wie bereits erwähnt, zeigt `head` standardmäßig nur die ersten 10 Zeilen der angegebenen Textdatei an. Mit Hilfe bestimmter Optionen ändert man dieses Verhalten. Weitere Informationen finden Sie in der Man Page des Befehls.

Es gibt daneben noch einen Befehl, der das Ende einer Datei anzeigt: `tail`. Standardmäßig wählt dieser Befehl die letzten 10 Zeilen aus und zeigt sie an. Aber wie bei `head` können Sie die Anzahl ändern (siehe die Man Page von `tail` für weitere Details).

NOTE Die Option `-f` zeigt die letzten Zeilen einer Datei an, während sie aktualisiert wird. Diese Funktion ist sehr nützlich, wenn Sie eine Datei wie `syslog` auf laufende Aktivitäten überwachen.

Der Befehl `wc` (“word count”) zählt standardmäßig die Zeilen, Wörter und Bytes einer Datei. Wie in der Übung gezeigt, bewirkt die Option `-w`, dass der Befehl nur die Wörter innerhalb der ausgewählten Zeilen zählt. Die häufigsten Optionen, die Sie mit diesem Befehl verwenden, sind `-l` (nur die Zeilen zählen) und `-c` (nur die Bytes zählen). Weitere Optionen des Befehls sowie weitere Informationen über `wc` finden Sie in der Man Page.

Geführte Übungen

1. Listen Sie den Inhalt Ihres aktuellen Verzeichnisses einschließlich Eigentümer und Berechtigungen auf und leiten Sie die Ausgabe in eine Datei namens `contents.txt` in Ihrem Heimatverzeichnis um.

2. Sortieren Sie den Inhalt der Datei `contents.txt` aus Ihrem aktuellen Verzeichnis und fügen Sie ihn an das Ende einer neuen Datei namens `contents-sorted.txt` an.

3. Zeigen Sie die letzten 10 Zeilen der Datei `/etc/passwd` an und leiten Sie sie in eine neue Datei im Verzeichnis `Documents` Ihres Benutzers um.

4. Zählen Sie die Anzahl der Wörter in der Datei `contents.txt` und hängen Sie die Ausgabe an das Ende einer Datei `field2.txt` in Ihrem Heimatverzeichnis an. Sie müssen sowohl die Eingabe- als auch die Ausgabeumleitung verwenden.

5. Zeigen Sie die ersten 5 Zeilen der Datei `/etc/passwd` an und sortieren Sie die Ausgabe alphabetisch umgekehrt.

6. Zählen Sie mit der zuvor erstellten Datei `contents.txt` die Anzahl der Zeichen der letzten 9 Zeilen.

7. Zählen Sie die Anzahl der Dateien namens `test` im Verzeichnis `/usr/share` und dessen Unterverzeichnissen. Hinweis: Jede Zeilenausgabe des Befehls `find` steht für eine Datei.

Offene Übungen

1. Wählen Sie das zweite Feld der Datei `contents.txt` aus und leiten Sie die Standardausgabe und Fehlerausgabe in eine andere Datei namens `field1.txt` um.

2. Löschen Sie mithilfe des Eingabeumleitungsoperators und des Befehl `tr` die Bindestriche (`-`) aus der Datei `contents.txt`.

3. Worin besteht der größte Vorteil, nur Fehler in eine Datei umzuleiten?

4. Ersetzen Sie alle aufeinanderfolgenden Leerzeichen in der alphabetisch sortierten Datei `contents.txt` durch ein einziges Leerzeichen.

5. Eliminieren Sie in einer Befehlszeile die aufeinanderfolgenden Leerzeichen (wie in der vorangehenden Übung), wählen Sie das neunte Feld aus und sortieren Sie es umgekehrt alphabetisch und nicht case-sensitiv. Wie viele Pipes mussten Sie verwenden?

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Arten der Umleitung
- Wie Sie die Umleitungsoperatoren verwenden
- Wie Sie Pipes zum Filtern der Befehlsausgabe verwenden

Befehle, die in dieser Lektion verwendet wurden:

`cut`
: Entfernt Abschnitte aus jeder Zeile einer Datei.

`cat`
: Zeigt Dateien an oder verknüpft sie.

`find`
: Sucht nach Dateien in einer Verzeichnishierarchie.

`less`
: Zeigt eine Datei an, so dass der Benutzer zeilenweise scrollen kann.

`more`
: Zeigt eine Datei seitenweise an.

`head`
: Zeigt die ersten 10 Zeilen einer Datei an.

`tail`
: Zeigt die letzten 10 Zeilen einer Datei an.

`sort`
: Sortiert Dateien.

`wc`
: Zählt standardmäßig die Zeilen, Wörter oder Bytes einer Datei.

Lösungen zu den geführten Übungen

1. Listen Sie den Inhalt Ihres aktuellen Verzeichnisses einschließlich Eigentümer und Berechtigungen auf und leiten Sie die Ausgabe in eine Datei namens `contents.txt` in Ihrem Heimatverzeichnis um.

```
$ ls -l > contents.txt
```

2. Sortieren Sie den Inhalt der Datei `contents.txt` aus Ihrem aktuellen Verzeichnis und fügen Sie ihn an das Ende einer neuen Datei namens `contents-sorted.txt` an.

```
$ sort contents.txt >> contents-sorted.txt
```

3. Zeigen Sie die letzten 10 Zeilen der Datei `/etc/passwd` an und leiten Sie sie in eine neue Datei im Verzeichnis `Documents` Ihres Benutzers um.

```
$ tail /etc/passwd > Documents/newfile
```

4. Zählen Sie die Anzahl der Wörter in der Datei `contents.txt` und hängen Sie die Ausgabe an das Ende einer Datei `field2.txt` in Ihrem Heimatverzeichnis an. Sie müssen sowohl die Eingabe- als auch die Ausgabeumleitung verwenden.

```
$ wc < contents.txt >> field2.txt
```

5. Zeigen Sie die ersten 5 Zeilen der Datei `/etc/passwd` an und sortieren Sie die Ausgabe alphabetisch umgekehrt.

```
$ head -n 5 /etc/passwd | sort -r
```

6. Zählen Sie mit der zuvor erstellten Datei `contents.txt` die Anzahl der Zeichen der letzten 9 Zeilen.

```
$ tail -n 9 contents.txt | wc -c
531
```

7. Zählen Sie die Anzahl der Dateien namens `test` im Verzeichnis `/usr/share` und dessen Unterverzeichnissen. Hinweis: Jede Zeilenausgabe des Befehls `find` steht für eine Datei.

```
$ find /usr/share -name test | wc -l
125
```

Lösungen zu den offenen Übungen

1. Wählen Sie das zweite Feld der Datei `contents.txt` aus und leiten Sie die Standardausgabe und Fehlerausgabe in eine andere Datei namens `field1.txt` um.

   ```
   $ cut -f 2 -d " " contents.txt &> field1.txt
   ```

2. Löschen Sie mithilfe des Eingabeumleitungsoperators und des Befehl `tr` die Bindestriche (`-`) aus der Datei `contents.txt`.

   ```
   $ tr -d "-" < contents.txt
   ```

3. Worin besteht der größte Vorteil, nur Fehler in eine Datei umzuleiten?

   Nur Fehler in eine Datei umzuleiten, kann bei der Führung einer Protokolldatei helfen, die regelmäßig überwacht wird.

4. Ersetzen Sie alle aufeinanderfolgenden Leerzeichen in der alphabetisch sortierten Datei `contents.txt` durch ein einziges Leerzeichen.

   ```
   $ sort contents.txt | tr -s " "
   ```

5. Eliminieren Sie in einer Befehlszeile die aufeinanderfolgenden Leerzeichen (wie in der vorangehenden Übung), wählen Sie das neunte Feld aus und sortieren Sie es umgekehrt alphabetisch und nicht case-sensitiv. Wie viele Pipes mussten Sie verwenden?

   ```
   $ cat contents.txt | tr -s " " | cut -f 9 -d " " | sort -fr
   ```

   Die Übung nutzt 3 Pipes – eine für jeden Filter.

3.2 Lektion 2

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	3 Die Macht der Befehlszeile
Lernziel:	3.2 Daten in Dateien suchen und extrahieren
Lektion:	2 von 2

Einführung

In dieser Lektion werden wir uns Werkzeuge zur Textmanipulation ansehen, die häufig von Systemadministratoren oder Programmen verwendet werden, um wiederkehrende Informationen automatisch zu überwachen oder zu identifizieren.

Suche innerhalb von Dateien mit `grep`

Das erste Werkzeug, das wir in dieser Lektion besprechen werden, ist der Befehl `grep`. `grep` ist die Abkürzung für "`global regular expression print`" und seine Hauptfunktion ist die Suche in Dateien nach einem angegebenen Muster. Der Befehl gibt die Zeile mit dem angegebenen Muster rot markiert aus.

```
$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
user:x:1001:1001:User,,,,:/home/user:/bin/bash
```

`grep` lässt sich, wie die meisten Befehle, über Optionen steuern. Hier sind die häufigsten:

`-i`

Die Suche ist case-insensitiv, ignoriert also Groß-/Kleinschreibung.

`-r`

Die Suche ist rekursiv (sie sucht in allen Dateien innerhalb des angegebenen Verzeichnisses und seiner Unterverzeichnisse).

`-c`

Die Suche zählt die Anzahl der Treffer.

`-v`

Kehrt die Übereinstimmung um und gibt Zeilen aus, die *nicht* dem Suchbegriff entsprechen.

`-E`

Schaltet erweiterte reguläre Ausdrücke ein (benötigt von einigen fortgeschritteneren Metazeichen wie `|`, `+` und `?`).

`grep` hat viele andere nützliche Optionen. Schauen Sie in die Man Page, um mehr darüber zu erfahren.

Reguläre Ausdrücke

Das zweite Werkzeug ist sehr leistungsfähig und dient der Beschreibung von Textteilen in Dateien, auch *reguläre Ausdrücke* genannt. Reguläre Ausdrücke sind äußerst nützlich bei der Extraktion von Daten aus Textdateien durch die Angabe von Mustern. Sie werden häufig in Skripten oder bei der Programmierung mit Hochsprachen wie Perl oder Python verwendet.

Bei der Arbeit mit regulären Ausdrücken ist es sehr wichtig zu beachten, dass *jedes Zeichen zählt* und dass das Muster mit dem Ziel geschrieben wird, eine bestimmte Zeichenfolge abzubilden, die als Zeichenkette oder *String* bezeichnet wird. Die meisten Muster verwenden die normalen ASCII-Zeichen, wie Buchstaben, Ziffern, Satzzeichen oder andere Symbole, aber sie können auch Unicode-Zeichen verwenden, um jede andere Art von Text abzubilden.

Die folgende Liste beschreibt die Metazeichen der regulären Ausdrücke, die zur Bildung der Muster verwendet werden:

`.`

Übereinstimmung eines einzelnen Zeichens (außer Zeilenumbruch).

`[abcABC]`

Übereinstimmung eines beliebigen Zeichens innerhalb der Klammern.

`[^abcABC]`
: Übereinstimmung eines beliebigen Zeichens außer denen innerhalb der Klammern.

`[a-z]`
: Übereinstimmung eines beliebigen Zeichens im angegebenen Bereich.

`[^a-z]`
: Übereinstimmung eines beliebigen Zeichens außer denen im angegebenen Bereich.

`sun|moon`
: Übereinstimmung mit einer der angegebenen Zeichenketten.

`^`
: Zeilenbeginn

`$`
: Zeilenende

Alle Funktionalitäten der regulären Ausdrücke lassen sich auch über `grep` implementieren. Im obigen Beispiel sehen Sie, dass das Wort nicht von doppelten Anführungszeichen umgeben ist. Um zu verhindern, dass die Shell das Metazeichen selbst interpretiert, wird empfohlen, komplexere Muster in doppelte Anführungszeichen (" ") zu setzen. Zu Übungszwecken verwenden wir bei der Implementierung regulärer Ausdrücke doppelte Anführungszeichen. Die anderen Anführungszeichen behalten ihre normale Funktionalität, wie in früheren Lektionen erläutert.

Die folgenden Beispiele verdeutlichen die Funktionalität der regulären Ausdrücke. Wir benötigen Daten in der Datei, daher hängt der nächste Befehlssatz einfach verschiedene Zeichenketten an die Datei `text.txt` an.

```
$ echo "aaabbb1" > text.txt
$ echo "abab2" >> text.txt
$ echo "noone2" >> text.txt
$ echo "class1" >> text.txt
$ echo "alien2" >> text.txt
$ cat text.txt
aaabbb1
abab2
noone2
class1
alien2
```

Das erste Beispiel ist eine Kombination aus der Suche in der Datei ohne und mit regulären Ausdrücken. Um reguläre Ausdrücke vollständig zu verstehen, ist es sehr wichtig, den Unterschied zu zeigen: Der erste Befehl sucht nach der genauen Zeichenkette an beliebiger Stelle in der Zeile, während der zweite Befehl nach Zeichenketten sucht, die eines der Zeichen zwischen den Klammern enthalten. Darum liefern die Befehle unterschiedliche Ergebnisse.

```
$ grep "ab" text.txt
aaabbb1
abab2
$ grep "[ab]" text.txt
aaabbb1
abab2
class1
alien2
```

Die nächsten Beispiele zeigen die Anwendung der Metazeichen für Zeilenanfang und Zeilenende. Es ist sehr wichtig, die beiden Zeichen an die richtige Stelle im Ausdruck zu setzen: Bei der Angabe des Zeilenanfangs muss das Metazeichen *vor* dem Ausdruck stehen, während es bei der Angabe des Zeilenendes *nach* dem Ausdruck stehen muss.

```
$ grep "^a" text.txt
aaabbb1
abab2
alien2
$ grep "2$" text.txt
abab2
noone2
alien2
```

Neben den zuvor erläuterten Metazeichen haben reguläre Ausdrücke weitere Metazeichen zur Wiederholung eines angegebenen Musters:

`*`

Null oder mehr Vorkommen des vorhergehenden Musters.

`+`

Ein oder mehr Vorkommen des vorhergehenden Musters.

`?`

Null oder ein Vorkommen des vorhergehenden Musters.

Als Beispiel für die Multiplikator-Metazeichen sucht der folgende Befehl nach einer Zeichenkette, die `ab`, ein einzelnes Zeichen und ein oder mehrere der zuvor gefundenen Zeichen enthält. Das Ergebnis zeigt, dass `grep` die Zeichenkette `aaabbb1` gefunden hat, die dem Teil `abbb` `entspricht, wie auch `abab2`. Da `+` ein Zeichen eines *erweiterten* regulären Ausdrucks ist, müssen wir dem Befehl `grep` die Option `-E` mitgeben.

```
$ grep -E "ab.+" text.txt
aaabbb1
abab2
```

Die meisten Metazeichen sind selbsterklärend, können aber bei der ersten Verwendung knifflig werden. Die vorherigen Beispiele stellen einen kleinen Teil der Funktionalität der regulären Ausdrücke dar. Probieren Sie alle genannten Metazeichen aus, um mehr darüber zu erfahren, wie sie funktionieren.

Geführte Übungen

Suchen Sie mit `grep` in der Datei `/usr/share/hunspell/en_US.dic` die Zeilen, die den folgenden Kriterien entsprechen:

1. Alle Zeilen, die die Zeichenfolge `cat` an beliebiger Stelle in der Zeile enthalten.

2. Alle Zeilen, die keines der folgenden Zeichen enthalten: `sawgtfixk`.

3. Alle Zeilen, die mit 3 beliebigen Buchstaben und der Zeichenfolge `dig` beginnen.

4. Alle Zeilen, die mit mindestens einem `e` enden.

5. Alle Zeilen, die eine der folgenden Zeichenfolgen enthalten: `org`, `kay` oder `tuna`.

6. Anzahl der Zeilen, die mit einem oder keinem `c` beginnen, gefolgt von der Zeichenkette `ati`.

Offene Übungen

1. Finden Sie den regulären Ausdruck, der mit den Wörtern in der Zeile “Einschließen” übereinstimmt und nicht mit denen in der Zeile “Ausschließen”:
   - Einschließen: `pot`, `spot`, `apot`

     Ausschließen: `potic`, `spots`, `potatoe`
   - Einschließen: `arp99`, `apple`, `zipper`

     Ausschließen: `zoo`, `arive`, `attack`
   - Einschließen: `arcane`, `capper`, `zoology`

     Ausschließen: `air`, `coper`, `zoloc`
   - Einschließen: `0th/pt`, `3th/tc`, `9th/pt`

     Ausschließen: `0/nm`, `3/nm`, `9/nm`
   - Einschließen: `Hawaii`, `Dario`, `Ramiro`

     Ausschließen: `hawaii`, `Ian`, `Alice`
2. Welcher andere nützliche Befehl wird häufig verwendet, um innerhalb von Dateien zu suchen. Welche zusätzlichen Funktionalitäten hat er?
3. Nutzen Sie ein Beispiel aus der vorangegangenen Lektion und versuchen Sie, mit Hilfe von `grep` nach einem bestimmten Muster in der Ausgabe des Befehls zu suchen.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Reguläre Ausdrücke und Metazeichen.
- Wie man Muster mit regulären Ausdrücken erstellt.
- Wie man in Dateien sucht.

Befehle, die in den Übungen verwendet werden:

`grep`
: Sucht nach Zeichen oder Zeichenketten in einer Datei.

Lösungen zu den geführten Übungen

Suchen Sie mit `grep` in der Datei `/usr/share/hunspell/en_US.dic` die Zeilen, die den folgenden Kriterien entsprechen:

1. Alle Zeilen, die die Zeichenfolge `cat` an beliebiger Stelle in der Zeile enthalten.

```
$ grep "cat" /usr/share/hunspell/en_US.dic
Alcatraz/M
Decatur/M
Hecate/M
...
```

2. Alle Zeilen, die keines der folgenden Zeichen enthalten: `sawgtfixk`.

```
$ grep -v "[sawgtfixk]" /usr/share/hunspell/en_US.dic
49269
0/nm
1/n1
2/nm
2nd/p
3/nm
3rd/p
4/nm
5/nm
6/nm
7/nm
8/nm
...
```

3. Alle Zeilen, die mit 3 beliebigen Buchstaben und der Zeichenfolge `dig` beginnen.

```
$ grep "^...dig" /usr/share/hunspell/en_US.dic
cardigan/SM
condign
predigest/GDS
...
```

4. Alle Zeilen, die mit mindestens einem `e` enden.

```
$ grep -E "e+$" /usr/share/hunspell/en_US.dic
Anglicize
Anglophobe
Anthropocene
...
```

5. Alle Zeilen, die eine der folgenden Zeichenfolgen enthalten: `org`, `kay` oder `tuna`.

```
$ grep -E "org|kay|tuna" /usr/share/hunspell/en_US.dic
Borg/SM
George/MS
Tokay/M
fortunate/UY
...
```

6. Anzahl der Zeilen, die mit einem oder keinem `c` beginnen, gefolgt von der Zeichenkette `ati`.

```
$ grep -cE "^c?ati" /usr/share/hunspell/en_US.dic
3
```

Lösungen zu den offenen Übungen

1. Finden Sie den regulären Ausdruck, der mit den Wörtern in der Zeile “Einschließen” übereinstimmt und nicht mit denen in der Zeile “Ausschließen”:
   - Einschließen: `pot`, `spot`, `apot`

     Ausschließen: `potic`, `spots`, `potatoe`

     Lösung: `pot$`
   - Einschließen: `arp99`, `apple`, `zipper`

     Ausschließen: `zoo`, `arive`, `attack`

     Lösung: `p+`
   - Einschließen: `arcane`, `capper`, `zoology`

     Ausschließen: `air`, `coper`, `zoloc`

     Lösung: `arc|cap|zoo`
   - Einschließen: `0th/pt`, `3th/tc`, `9th/pt`

     Ausschließen: `0/nm`, `3/nm`, `9/nm`

     Lösung: `[0-9]th.+`
   - Einschließen: `Hawaii`, `Dario`, `Ramiro`

     Ausschließen: `hawaii`, `Ian`, `Alice`

     Lösung: `^[A-Z]a.*i+`
2. Welcher andere nützliche Befehl wird häufig verwendet, um innerhalb von Dateien zu suchen. Welche zusätzlichen Funktionalitäten hat er?

   Der Befehl `sed`. Er kann Zeichen oder Zeichenfolgen in einer Datei suchen und ersetzen.
3. Nutzen Sie ein Beispiel aus der vorangegangenen Lektion und versuchen Sie, mit Hilfe von `grep` nach einem bestimmten Muster in der Ausgabe des Befehls zu suchen.

   Ich nehme eine der Lösungen aus den Offenen Übungen und suche nach der Zeile, die als Gruppenberechtigungen lesen, schreiben und ausführen hat. Ihre Antwort kann anders lauten,

abhängig davon, welchen Befehl Sie gewählt und welches Muster Sie erstellt haben.

```
$ cat contents.txt | tr -s " " | grep "^....rwx"
```

Diese Übung soll Ihnen zeigen, dass `grep` auch Eingaben von verschiedenen Befehlen erhalten kann und dass es beim Filtern generierter Informationen hilft.

3.3 Von Befehlen zum Skript

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 3.3

Gewichtung

4

Hauptwissensgebiete

- Grundlagen von Shell-Skripten
- Wissen über die gängigen Texteditoren (vi and nano)

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- `#!` (shebang)
- `/bin/bash`
- Variablen
- Argumente
- for-Loops
- `echo`
- Exit-Status

3.3 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	3 Die Macht der Befehlszeile
Lernziel:	3.3 Von Befehlen zum Skript
Lektion:	1 von 2

Einführung

Wir haben bisher gelernt, Befehle von der Shell aus auszuführen, aber wir können auch Befehle in eine Datei schreiben und diese dann ausführbar machen. Wenn die Datei ausgeführt wird, werden diese Befehle nacheinander ausgeführt. Diese ausführbaren Dateien heißen *Skripte* und sind ein unerlässliches Werkzeug für jeden Linux-Systemadministrator. Im Grunde können wir Bash sowohl als Programmiersprache wie auch als Shell betrachten.

Ausgabe anzeigen

Beginnen wir mit einem Befehl, den Sie vielleicht schon in früheren Lektionen gesehen haben: `echo` gibt ein Argument in die Standardausgabe aus.

```
$ echo "Hello World!"
Hello World!
```

Nun nutzen wir die Dateiumleitung, um diesen Befehl in eine neue Datei namens `new_script` zu schreiben.

```
$ echo 'echo "Hello World!"' > new_script
$ cat new_script
echo "Hello World!"
```

Die Datei `new_script` enthält nun den gleichen Befehl wie zuvor.

Ein Skript ausführbar machen

Gehen wir die notwendigen Schritte durch, damit diese Datei das tut, was wir erwarten. Der erste Gedanke eines Benutzers könnte sein, einfach den Namen des Skripts einzugeben, so wie er den Namen anderer Befehle eingibt:

```
$ new_script
/bin/bash: new_script: command not found
```

Wir wissen, dass `new_script` an unserem aktuellen Standort existiert, aber beachten Sie, dass die Fehlermeldung uns nicht sagt, dass die *Datei* nicht existiert — sie sagt uns, der *Befehl* existiert nicht.

Befehle und `PATH`

Wenn wir beispielsweise den Befehl `ls` in die Shell eingeben, führen wir eine Datei namens `ls` aus, die in unserem Dateisystem existiert. Sie können dies mit `which` zeigen:

```
$ which ls
/bin/ls
```

Es würde schnell lästig werden, jedes Mal den absoluten Pfad von `ls` einzugeben, wenn wir den Inhalt eines Verzeichnisses betrachten wollen; darum hat Bash eine *Umgebungsvariable* (*environment variable*), die alle Verzeichnisse enthält, in denen wir die ausführbaren Befehle finden. Schauen Sie sich den Inhalt dieser Variable mit `echo` an:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sn
ap/bin
```

An jedem dieser Orte, durch Doppelpunkte (`:`) getrennt, erwartet die Shell Befehle zu finden. Sie werden dort `/bin` sehen, aber es dürfte sicher sein, dass unser aktueller Standort nicht aufgeführt

ist. Die Shell wird in jedem dieser Verzeichnisse nach `new_script` suchen, aber sie wird es nicht finden und daher den oben genannten Fehler anzeigen.

Es gibt drei Lösungen für dieses Problem: Wir können `new_script` in eines der `PATH` -Verzeichnisse verschieben, wir können unser aktuelles Verzeichnis zu `PATH` hinzufügen, oder wir ändern die Art und Weise, wie wir das Skript aufrufen. Die letzte Lösung ist die einfachste, denn sie verlangt lediglich, dass wir den *aktuellen Standort* angeben, wenn wir das Skript mit einem Punktstrich (`./`) aufrufen.

```
$ ./new_script
/bin/bash: ./new_script: Permission denied
```

Die Fehlermeldung hat sich geändert, was darauf hindeutet, dass wir einige Fortschritte gemacht haben.

Ausführungsrechte

Die erste Untersuchung, die ein Benutzer in diesem Fall durchführen sollte, ist die Verwendung von `ls -l`, um sich die Datei anzusehen:

```
$ ls -l new_script
-rw-rw-r-- 1 user user 20 Apr 30 12:12 new_script
```

Wir sehen, dass die Berechtigungen für diese Datei standardmäßig auf `664` gesetzt sind. Wir haben für diese Datei noch keine *Ausführungsrechte* gesetzt.

```
$ chmod +x new_script
$ ls -l new_script
-rwxrwxr-x 1 user user 20 Apr 30 12:12 new_script
```

Dieser Befehl hat *allen* Benutzern Ausführungsrechte gewährt. Beachten Sie, dass dies ein Sicherheitsrisiko darstellen könnte, aber im Moment ist dies eine akzeptable Berechtigungsstufe.

```
$ ./new_script
Hello World!
```

Wir sind nun in der Lage, unser Skript auszuführen.

Definition des Interpreters

Wie wir gezeigt haben, konnten wir einfach Text in eine Datei schreiben, die Datei ausführbar machen und ausführen. `new_script` ist funktional immer noch eine normale Textdatei, aber wir haben es geschafft, sie von Bash interpretieren zu lassen. Aber was ist, wenn sie in Perl oder Python geschrieben ist?

Es ist sehr empfehlenswert, den Typ des Interpreters anzugeben, den wir in der ersten Zeile eines Skripts verwenden möchten. Diese Zeile wird *Bang Line* oder häufiger *Shebang* genannt und zeigt dem System an, wie diese Datei ausgeführt werden soll. Da wir Bash lernen, werden wir den absoluten Pfad zu unserer ausführbaren Bash-Datei verwenden, wiederum mit `which`:

```
$ which bash
/bin/bash
```

Unser Shebang beginnt mit einem Hashzeichen und einem Ausrufezeichen, gefolgt vom absoluten Pfad oben. Öffnen wir `new_script` in einem Texteditor und fügen den Shebang ein. Nutzen wir die Gelegenheit, einen *Kommentar* in unser Skript einzufügen, Kommentare werden vom Interpreten ignoriert und für andere Benutzer geschrieben, die Ihr Skript verstehen wollen.

```
#!/bin/bash

# This is our first comment. It is also good practice to document all scripts.

echo "Hello World!"
```

Wir werden noch eine weitere Änderung am Dateinamen vornehmen: Wir speichern diese Datei als `new_script.sh`. Das Dateisuffix `.sh` ändert die Ausführung der Datei in keiner Weise, es ist eine Konvention, dass Bash-Skripte mit `.sh` oder `.bash` gekennzeichnet werden, um sie leichter zu identifizieren, so wie Python-Skripte normalerweise mit dem Suffix `.py` identifiziert werden.

Häufig genutzte Texteditoren

Linux-Anwender müssen oft in einer Umgebung arbeiten, in der grafische Texteditoren nicht zur Verfügung stehen, weshalb es dringend empfohlen wird, sich zumindest mit der Bearbeitung von Textdateien über die Befehlszeile vertraut zu machen. Zwei der gängigsten Texteditoren sind `vi` und `nano`.

vi

`vi` ist ein ehrwürdiger Texteditor und wird standardmäßig auf fast jedem Linux-System installiert. Von `vi` gibt es einen Klon namens *vi IMproved* oder `vim`, der einige Funktionen hinzufügt, aber die Oberfläche von `vi` beibehält. Während die Arbeit mit `vi` für einen neuen Benutzer entmutigend ist, ist der Editor verbreitet und beliebt bei Benutzern, die seine vielen Funktionen kennen.

Der wichtigste Unterschied zwischen `vi` und Anwendungen wie Notepad besteht darin, dass `vi` drei verschiedene Modi hat: Beim Start werden die Tasten `H`, `J`, `K` und `L` zum Navigieren und nicht zum Tippen verwendet. `I` können Sie in diesem Navigationsmodus drücken, um in den *Einfügemodus* zu gelangen. Darin können Sie normal tippen. Um den Einfügemodus zu verlassen, drücken Sie `Esc`, um zum *Navigationsmodus* zurückzukehren. Im Navigationsmodus können Sie `:` drücken, um in den *Befehlsmodus* zu gelangen. In diesem Modus können Sie speichern, löschen, beenden oder Optionen ändern.

Während `vi` eine Lernkurve hat, erlauben die verschiedenen Modi mit der Zeit einem versierten Benutzer, effizienter zu werden als mit anderen Editoren.

nano

`nano` ist ein neueres Werkzeug, das einfacher zu bedienen ist als `vi`. `nano` hat keine unterschiedlichen Modi; ein Benutzer kann beim Start mit der Eingabe beginnen und verwendet `Strg`, um auf die am unteren Bildschirmrand gedruckten Werkzeuge zuzugreifen.

```
                        [ Welcome to nano.  For basic help, type Ctrl+G. ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

Texteditoren sind eine Frage der persönlichen Präferenz. Der Editor, den Sie verwenden, wird keinen Einfluss auf diese Lektion haben, aber sich in Zukunft mit einem oder mehreren Texteditoren vertraut zu machen, wird sich auszahlen.

Variablen

Variablen sind ein wichtiger Bestandteil jeder Programmiersprache, und Bash ist da nicht anders. Wenn Sie eine neue Sitzung vom Terminal aus starten, setzt die Shell bereits einige Variablen, wie z.B. die Variable `PATH`. Man nennt sie *Umgebungsvariablen* (*environment variables*), da sie in der Regel Eigenschaften unserer Shell-Umgebung definieren. Sie können Umgebungsvariablen ändern und hinzufügen, aber vorerst konzentrieren wir uns auf das Setzen von Variablen in unserem Skript.

Wir werden unser Skript wie folgt verändern:

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username=Carol

echo "Hello $username!"
```

In diesem Fall haben wir eine *Variable* namens `username` erstellt und haben ihr den *Wert* `Carol` zugewiesen. Beachten Sie, dass zwischen dem Variablennamen, dem Gleichheitszeichen und dem zugewiesenen Wert keine Leerzeichen stehen.

In der nächsten Zeile haben wir den Befehl `echo` mit der Variablen verwendet, aber vor dem Variablennamen steht ein Dollarzeichen (`$`), was wichtig ist, da es der Shell anzeigt, dass wir `username` als Variable und nicht nur als normales Wort behandeln wollen. Indem wir `username` in unseren Befehl eingeben, geben wir an, dass wir eine *Substitution* durchführen wollen, indem wir den *Namen* einer Variablen durch den dieser Variablen zugeordneten *Wert* ersetzen.

Wenn wir das neue Skript ausführen, erhalten wir diese Ausgabe:

```
$ ./new_script.sh
Hello Carol!
```

- Variablennamen dürfen nur alphanumerische Zeichen oder Unterstriche enthalten und sind case-sensitiv. `username` und `Username` werden als unterschiedliche Variablen behandelt.
- Die Variablenersetzung kann auch das Format `${username}` haben, mit dem Zusatz des `{ }`. Dies ist ebenfalls akzeptabel.
- Variablen in Bash haben einen *impliziten Typ* und werden als Zeichenketten betrachtet, was bedeutet, dass die Ausführung mathematischer Funktionen in Bash komplizierter ist als in anderen Programmiersprachen wie etwa C/C++:

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username=Carol
x=2
y=4
```

```
z=$x+$y
echo "Hello $username!"
echo "$x + $y"
echo "$z"
```

```
$ ./new_script.sh
Hello Carol!
2 + 4
2+4
```

Verwendung von Anführungszeichen bei Variablen

Nehmen wir die folgende Änderung am Wert unserer Variable `username` vor:

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username=Carol Smith

echo "Hello $username!"
```

Die Ausführung dieses Skripts führt zu einem Fehler:

```
$ ./new_script.sh
./new_script.sh: line 5: Smith: command not found
Hello !
```

Beachten Sie, dass Bash ein Interpreter ist, und als solcher *interpretiert* es unser Skript Zeile für Zeile. In diesem Fall interpretiert es `username=Carol` korrekt, um eine Variable `username` mit dem Wert `Carol` zu setzen. Aber dann interpretiert es das Leerzeichen als Ende dieser Zuordnung und `Smith` als den Namen eines Befehls. Um auch das Leerzeichen und den Namen `Smith` als Wert unserer Variable hinzuzufügen, setzen wir doppelte Anführungszeichen (`"`) um den Namen.

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username="Carol Smith"
```

```
echo "Hello $username!"
```

```
$ ./new_script.sh
Hello Carol Smith!
```

Eine wichtige Sache, die in Bash zu beachten ist: Doppelte und einfache Anführungszeichen (`'`) verhalten sich sehr unterschiedlich. Doppelte Anführungszeichen gelten als “schwach”, weil sie es dem Interpreter ermöglichen, eine Ersetzung innerhalb der Anführungszeichen durchzuführen. Einfache Anführungszeichen gelten als “stark”, weil sie jede Ersetzung verhindern:

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username="Carol Smith"

echo "Hello $username!"
echo 'Hello $username!'
```

```
$ ./new_script.sh
Hello Carol Smith!
Hello $username!
```

Im zweiten Befehl `echo` wurde verhindert, dass der Interpreter `$username` durch `Carol Smith` ersetzt, so dass die Ausgabe wörtlich genommen wird.

Argumente

Sie sind bereits mit der Verwendung von Argumenten in den Linux-Kerndienstprogrammen vertraut, z.B. enthält `rm testfile` sowohl die ausführbare Datei `rm` als auch ein Argument `testfile`. Argumente können bei der Ausführung an das Skript übergeben werden und ändern das Verhalten des Skripts. Sie sind einfach zu implementieren.

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username=$1
```

```
echo "Hello $username!"
```

Anstatt `username` direkt im Skript einen Wert zuzuweisen, weisen wir ihm den Wert einer neuen Variablen `$1` zu. Diese verweist auf den Wert des *ersten Arguments*.

```
$ ./new_script.sh Carol
Hello Carol!
```

Die ersten neun Argumente werden auf diese Weise behandelt. Es gibt Möglichkeiten, mehr als neun Argumente zu verarbeiten, aber das liegt außerhalb des Rahmens dieser Lektion. Wir werden ein Beispiel mit nur zwei Argumenten zeigen:

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username1=$1
username2=$2
echo "Hello $username1 and $username2!"
```

```
$ ./new_script.sh Carol Dave
Hello Carol and Dave!
```

Bei der Verwendung von Argumenten gibt es eine wichtige Überlegung: Im obigen Beispiel gibt es zwei Argumente `Carol` und `Dave`, die jeweils `$1` und `$2` zugeordnet sind. Fehlt beispielsweise das zweite Argument, wirft die Shell keinen Fehler, der Wert von `$2` ist einfach *null* oder gar nichts.

```
$ ./new_script.sh Carol
Hello Carol and !
```

In unserem Fall wäre es eine gute Idee, unserem Skript eine gewisse Logik beizufügen, damit verschiedene *Bedingungen* die *Ausgabe* beeinflussen. Wir beginnen mit der Einführung einer weiteren hilfreichen Variablen und gehen dann zur Erstellung von *if-Anweisungen* über.

Anzahl der Argumente zurückgeben

Während Variablen wie `$1` und `$2` den Wert von Positionsargumenten enthalten, enthält eine

andere Variable `$#` die *Anzahl der Argumente*.

```
#!/bin/bash

# This is our first comment. It is also good practice to comment all scripts.

username=$1

echo "Hello $username!"
echo "Number of arguments: $#."
```

```
$ ./new_script.sh Carol Dave
Hello Carol!
Number of arguments: 2.
```

Bedingungslogik (Conditional Logic)

Bedingungslogik ist in der Programmierung ein weites Feld und wird in dieser Lektion nicht ausführlich behandelt. Wir konzentrieren uns auf die *Syntax* bedingter Anweisungen in Bash, die sich von den meisten anderen Programmiersprachen unterscheidet.

Beginnen wir mit einem einfachen Skript, das eine Begrüßung an einen einzelnen Benutzer ausgeben soll. Bei etwas anderem als einem Benutzer, soll es eine Fehlermeldung ausgeben.

- Die Bedingung, die wir testen, ist die Anzahl der Benutzer, die in der Variablen `$#` enthalten ist. Wir würden gerne wissen, ob `$#` den Wert `1` hat.
- Wenn die Bedingung *wahr* (*true*) ist, ist die *Aktion*, die wir ausführen, die Begrüßung des Benutzers.
- Wenn die Bedingung *falsch* (*false*) ist, geben wir eine Fehlermeldung aus.

Nun, da die Logik klar ist, wenden wir uns der *Syntax* zu, die zur Implementierung dieser Logik erforderlich ist.

```
#!/bin/bash

# A simple script to greet a single user.

if [ $# -eq 1 ]
then
  username=$1
```

```
  echo "Hello $username!"
else
  echo "Please enter only one argument."
fi
echo "Number of arguments: $#."
```

Die Bedingungslogik liegt zwischen `if` und `fi`. Die zu testende Bedingung befindet sich zwischen eckigen Klammern `[ ]`, und die Maßnahmen, die zu ergreifen sind, wenn die Bedingung wahr ist, werden nach `then` angezeigt. Beachten Sie die Leerzeichen zwischen den eckigen Klammern und der enthaltenen Logik. Das Weglassen dieser Leerzeichen führt zu Fehlern.

Dieses Skript gibt entweder unsere Begrüßung *oder* die Fehlermeldung aus. Aber es gibt immer die Zeile `Number of arguments` aus.

```
$ ./new_script.sh
Please enter only one argument.
Number of arguments: 0.
$ ./new_script.sh Carol
Hello Carol!
Number of arguments: 1.
```

Beachten Sie die `if`-Anweisung: Wir haben mit `-eq` einen *numerischen Vergleich* durchgeführt. In diesem Fall testen wir, ob der Wert von `$#` *gleich* eins ist. Die anderen Vergleiche, die wir durchführen können, sind:

`-ne`
: Nicht gleich

`-gt`
: Größer als

`-ge`
: Größer oder gleich

`-lt`
: Kleiner als

`-le`
: Kleiner oder gleich

Geführte Übungen

1. Der Benutzer gibt Folgendes in seine Shell ein:

```
$ PATH=~/scripts
$ ls
Command 'ls' is available in '/bin/ls'
The command could not be located because '/bin' is not included in the PATH environment
variable.
ls: command not found
```

   - Was hat der Benutzer gemacht?

   - Welcher Befehl kombiniert den aktuellen Wert von `PATH` mit dem neuen Verzeichnis `~/scripts`?

2. Betrachten Sie das folgende Skript. Beachten Sie, dass es `elif` verwendet, um nach einer zweiten Bedingung zu suchen:

```
>  /!bin/bash

> fruit1 = Apples
> fruit2 = Oranges

  if [ $1 -lt $# ]
  then
    echo "This is like comparing $fruit1 and $fruit2!"
> elif [$1 -gt $2 ]
  then
>   echo '$fruit1 win!'
  else
>   echo "Fruit2 win!"
> done
```

   - Die mit einem `>` markierten Zeilen enthalten Fehler. Beheben Sie die Fehler.

3. Was wird in den folgenden Situationen ausgegeben?

```
$ ./guided1.sh 3 0
```

```
$ ./guided1.sh 2 4
```

```
$ ./guided1.sh 0 1
```

Offene Übungen

1. Schreiben Sie ein einfaches Skript, das überprüft, ob genau zwei Argumente übergeben werden. Wenn ja, drucken Sie die Argumente in umgekehrter Reihenfolge. Betrachten Sie dieses Beispiel (Hinweis: Ihr Code kann anders aussehen, sollte aber zur gleichen Ausgabe führen):

```
if [ $1 == $number ]
then
  echo "True!"
fi
```

2. Dieser Code ist korrekt, aber es handelt sich nicht um einen Zahlenvergleich. Verwenden Sie eine Internetsuche, um herauszufinden, worin sich dieser Code von der Verwendung von `-eq` unterscheidet.

3. Es gibt eine Umgebungsvariable, die das aktuelle Verzeichnis ausgibt. Verwenden Sie `env`, um den Namen dieser Variable zu ermitteln.

4. Schreiben Sie unter Verwendung dessen, was Sie in den Fragen 2 und 3 gelernt haben, ein kurzes Skript, das ein Argument akzeptiert. Wenn ein Argument übergeben wird, überprüfen Sie, ob dieses Argument mit dem Namen des aktuellen Verzeichnisses übereinstimmt. Wenn ja, drucken Sie `ja`. Andernfalls drucken Sie `nein`.

Zusammenfassung

In diesem Abschnitt haben Sie gelernt:

- Wie man einfache Skripte erstellt und ausführt
- Wie man einen Shebang verwendet, um einen Interpreter anzugeben
- Wie man Variablen in Skripten setzt und verwendet
- Wie man Argumente in Skripten behandelt
- Wie man `if`-Anweisungen konstruiert

- Wie man Zahlen mit numerischen Operatoren vergleicht

Befehle, die in den Übungen verwendet werden:

`echo`
: Druckt eine Zeichenkette in die Standardausgabe.

`env`
: Druckt alle Umgebungsvariablen in die Standardausgabe.

`which`
: Gibt den absoluten Pfad eines Befehls aus.

`chmod`
: Ändert die Berechtigungen einer Datei.

Spezielle Variablen, die in den Übungen verwendet werden:

`$1, $2, … $9`
: Enthält Positionsargumente, die an das Skript übergeben werden.

`$#`
: Enthält die Anzahl der an das Skript übergebenen Argumente.

`$PATH`
: Enthält die Verzeichnisse mit ausführbaren Dateien, die vom System verwendet werden.

Operatoren, die in den Übungen eingesetzt werden:

`-ne`
: Nicht gleich

`-gt`
: Größer als

`-ge`
: Größer oder gleich

`-lt`
: Kleiner als

`-le`

Kleiner oder gleich

Lösungen zu den geführten Übungen

1. Der Benutzer gibt Folgendes in seine Shell ein:

```
$ PATH=~/scripts
$ ls
Command 'ls' is available in '/bin/ls'
The command could not be located because '/bin' is not included in the PATH environment
variable.
ls: command not found
```

   - Was hat der Benutzer gemacht?

     Der Benutzer hat den Inhalt von `PATH` mit dem Verzeichnis `~/scripts` überschrieben. Der Befehl `ls` kann nicht mehr gefunden werden, da er nicht in `PATH` enthalten ist. Beachten Sie, dass diese Änderung nur die aktuelle Sitzung betrifft. Wenn Sie sich aus- und wieder einloggen, sind die Änderungen rückgängig gemacht.

   - Welcher Befehl kombiniert den aktuellen Wert von `PATH` mit dem neuen Verzeichnis `~/scripts`?

     `PATH=$PATH:~/scripts`

2. Betrachten Sie das folgende Skript. Beachten Sie, dass es `elif` verwendet, um nach einer zweiten Bedingung zu suchen:

```
>  /!bin/bash

> fruit1 = Apples
> fruit2 = Oranges

  if [ $1 -lt $# ]
  then
    echo "This is like comparing $fruit1 and $fruit2!"
> elif [$1 -gt $2 ]
  then
>   echo '$fruit1 win!'
  else
>   echo "Fruit2 win!"
> done
```

- The lines marked with a `>` contain errors. Fix the errors.

```
#!/bin/bash

fruit1=Apples
fruit2=Oranges

uif [ $1 -lt $# ]
then
  echo "This is like comparing $fruit1 and $fruit2!"
elif [ $1 -gt $2 ]
then
  echo "$fruit1 win!"
else
  echo "$fruit2 win!"
fi
```

3. Was wird in den folgenden Situationen ausgegeben?

```
$ ./guided1.sh 3 0
```

`Apples win!`

```
$ ./guided1.sh 2 4
```

`Oranges win!`

```
$ ./guided1.sh 0 1
```

`This is like comparing Apples and Oranges!`

Lösungen zu den offenen Übungen

1. Schreiben Sie ein einfaches Skript, das überprüft, ob genau zwei Argumente übergeben werden. Wenn ja, drucken Sie die Argumente in umgekehrter Reihenfolge. Betrachten Sie dieses Beispiel (Hinweis: Ihr Code kann anders aussehen, sollte aber zur gleichen Ausgabe führen):

```
if [ $1 == $number ]
then
  echo "True!"
fi
```

```
#!/bin/bash

if [ $# -ne 2 ]
then
  echo "Error"
else
  echo "$2 $1"
fi
```

2. Dieser Code ist korrekt, aber es handelt sich nicht um einen Zahlenvergleich. Verwenden Sie eine Internetsuche, um herauszufinden, worin sich dieser Code von der Verwendung von `-eq` unterscheidet.

   Mit `==` werden *Strings* verglichen. Das heißt, wenn die Zeichen beider Variablen exakt übereinstimmen, dann ist die Bedingung wahr.

`abc == abc`	*true*
`abc == ABC`	*false*
`1 == 1`	*true*
`1+1 == 2`	*false*

   String-Vergleiche führen zu unerwartetem Verhalten, wenn Sie auf Zahlen testen.

3. Es gibt eine Umgebungsvariable, die das aktuelle Verzeichnis ausgibt. Verwenden Sie `env`, um den Namen dieser Variable zu ermitteln.

   `PWD`

4. Schreiben Sie unter Verwendung dessen, was Sie in den Fragen 2 und 3 gelernt haben, ein kurzes Skript, das ein Argument akzeptiert. Wenn ein Argument übergeben wird, überprüfen Sie, ob dieses Argument mit dem Namen des aktuellen Verzeichnisses übereinstimmt. Wenn ja, drucken Sie `ja`. Andernfalls drucken Sie `nein`.

```
#!/bin/bash

if [ "$1" == "$PWD" ]
then
  echo "yes"
else
  echo "no"
fi
```

3.3 Lektion 2

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	3 Die Macht der Befehlszeile
Lernziel:	3.3 Von Befehlen zum Skript
Lektion:	2 von 2

Einführung

Im letzten Abschnitt haben wir dieses einfache Beispiel verwendet, um das Bash-Skripting zu demonstrieren:

```
#!/bin/bash

# A simple script to greet a single user.

if [ $# -eq 1 ]
then
  username=$1

  echo "Hello $username!"
else
  echo "Please enter only one argument."
fi
echo "Number of arguments: $#."
```

- Alle Skripte sollten mit einem *Shebang* beginnen, der den Pfad zum Interpreter definiert.
- Alle Skripte sollten Kommentare enthalten, um ihre Verwendung zu beschreiben.
- Dieses spezielle Skript arbeitet mit einem Argument, das beim Aufruf an das Skript übergeben wird.
- Dieses Skript enthält eine *if-Anweisung*, die die Bedingungen einer eingebauten Variablen `$#` testet. Diese Variable wird auf die Anzahl der Argumente gesetzt.
- Wenn die Anzahl der an das Skript übergebenen Argumente gleich 1 ist, dann wird der Wert des ersten Arguments an eine neue Variable namens `username` übergeben und das Skript gibt einen Gruß aus. Andernfalls erscheint eine Fehlermeldung.
- Schließlich gibt das Skript die Anzahl der Argumente aus. Dies ist für die Fehlersuche nützlich.

Dies ist ein nützliches Beispiel, um einige weitere Funktionen von Bash-Skripting zu erklären.

Exit Codes

Sie werden feststellen, dass unser Skript zwei mögliche Zustände hat: Entweder es druckt `"Hello <user>!"` oder es gibt eine Fehlermeldung aus, was für viele unserer Kern-Utilities ganz normal ist. Denken Sie an `cat`, mit dem Sie zweifellos sehr vertraut werden.

Lassen Sie uns eine erfolgreiche Verwendung von `cat` mit einer Situation vergleichen, in der es fehlschlägt. Erinnern wir uns, dass unser obiges Beispiel ein Skript namens `new_script.sh` ist.

```
$ cat -n new_script.sh

     1  #!/bin/bash
     2
     3  # A simple script to greet a single user.
     4
     5  if [ $# -eq 1 ]
     6  then
     7    username=$1
     8
     9    echo "Hello $username!"
    10  else
    11    echo "Please enter only one argument."
    12  fi
    13  echo "Number of arguments: $#."
```

Dieser Befehl ist erfolgreich, und Sie werden feststellen, dass das `-n`-Flag auch Zeilennummern

gedruckt hat, die sehr hilfreich beim Debuggen von Skripten sind, aber bitte beachten Sie, dass sie *nicht* Teil des Skripts sind.

Jetzt werden wir den Wert einer neuen eingebauten Variablen `$?` überprüfen. Im Moment beachten Sie einfach die Ausgabe:

```
$ echo $?
0
```

Betrachten wir nun eine Situation, in der `cat` fehlschlägt. Zuerst erhalten wir eine Fehlermeldung und überprüfen dann den Wert von `$?`.

```
$ cat -n dummyfile.sh
cat: dummyfile.sh: No such file or directory
$ echo $?
1
```

Die Erklärung für dieses Verhalten ist folgende: Jede Ausführung des Dienstprogramms `cat` gibt einen *Exit Code* zurück. Ein Exit Code sagt uns, ob der Befehl erfolgreich war oder ein Fehler auftrat. Ein Exit Code von *null* zeigt an, dass der Befehl erfolgreich abgeschlossen wurde, was für fast jeden Linux-Befehl gilt, mit dem Sie arbeiten. Jeder andere Exit Code weist auf einen Fehler hin. Der Exit Code des *letzten auszuführenden Befehls* wird in der Variablen `$?` gespeichert.

Exit Codes werden normalerweise nicht von menschlichen Benutzern gesehen, aber sie sind sehr nützlich beim Schreiben von Skripten. Stellen Sie sich ein Skript vor, bei dem wir Dateien auf ein entferntes Netzlaufwerk kopieren. Es gibt viele Möglichkeiten, wie die Kopieraufgabe fehlgeschlagen sein kann: Unser lokaler Rechner ist nicht mit dem Netzwerk verbunden oder das Remote-Laufwerk ist voll. Indem wir den Exit Code unseres Kopierprogramms überprüfen, können wir den Benutzer auf Probleme beim Ausführen des Skripts aufmerksam machen.

Es ist gute Praxis, Exit Codes zu implementieren, also werden wir das jetzt tun. Wir haben zwei Pfade in unserem Skript: Erfolg und Misserfolg. Wir nutzen *null*, um Erfolg anzuzeigen, und *eins* für den Misserfolg.

```
1  #!/bin/bash
2
3  # A simple script to greet a single user.
4
5  if [ $# -eq 1 ]
6  then
```

```
 7    username=$1
 8
 9    echo "Hello $username!"
10    exit 0
11  else
12    echo "Please enter only one argument."
13    exit 1
14  fi
15  echo "Number of arguments: $#."
```

```
$ ./new_script.sh Carol
Hello Carol!
$ echo $?
0
```

Beachten Sie, dass der Befehl `echo` in Zeile 15 vollständig ignoriert wurde und das Skript mit `exit` sofort beendet wird, so dass diese Zeile nie gefunden wird.

Behandlung mehrerer Argumente

Bisher kann unser Skript nur einen einzigen Benutzernamen auf einmal verarbeiten, eine beliebige Anzahl von Argumenten außer einem wirft einen Fehler. Wir wollen herausfinden, wie wir dieses Skript vielseitiger gestalten können.

Ein Benutzer könnte instinktiv dazu übergehen, mehr Positionsvariablen wie `$2`, `$3` usw. zu verwenden. Leider können wir aber die Anzahl der Argumente, die ein Benutzer benötigt, nicht vorhersehen. Um dieses Problem zu lösen, ist es hilfreich, mehr eingebaute Variablen einzuführen.

Wir werden die Logik unseres Skripts ändern: Null Argumente sollen einen Fehler werfen, aber eine beliebige Anzahl von Argumenten sollte erfolgreich sein. Dieses neue Skript wird `friendly2.sh` heißen.

```
1  #!/bin/bash
2
3  # a friendly script to greet users
4
5  if [ $# -eq 0 ]
6  then
7    echo "Please enter at least one user to greet."
8    exit 1
```

```
 9  else
10    echo "Hello $@!"
11    exit 0
12  fi
```

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol Dave Henry!
```

Es gibt zwei eingebaute Variablen, die alle an das Skript übergebenen Argumente enthalten: `$@` und `$*`. Meist verhalten sich beide gleich, Bash wird die Argumente *parsen* und jedes Argument trennen, wenn es auf ein Leerzeichen stößt. Der Inhalt von `$@` sieht also so aus:

`0`	`1`	`2`
`Carol`	`Dave`	`Henry`

Wenn Sie mit anderen Programmiersprachen vertraut sind, erkennen Sie diese Art von Variable vielleicht als *Array*. Arrays sind in Bash einfach zu erstellen, indem Sie Leerzeichen zwischen Elementen wie der Variablen `FILES` im Skript `arraytest` unten setzen:

```
FILES="/usr/sbin/accept /usr/sbin/pwck /usr/sbin/chroot"
```

Es enthält eine Liste mit mehreren Elementen. Das ist bisher nicht sehr hilfreich, da wir noch keine Möglichkeit eingeführt haben, diese Elemente individuell zu behandeln.

For-Schleifen

Kommen wir noch einmal zu dem zuvor gezeigten Beispiel `arraytest`. Hier legen wir ein eigenes Array namens `FILES` an. Wir brauchen eine Methode, um diese Variable zu “entpacken” und nacheinander auf jeden einzelnen Wert zugreifen zu können. Dafür nutzem wir eine Struktur namens *for-Schleife*, die es in allen Programmiersprachen gibt. Es gibt zwei Variablen, auf die wir zugreifen werden: eine ist der Bereich, und die andere ist für den individuellen Wert, an dem wir gerade arbeiten. Dies ist das gesamt Skript:

```
#!/bin/bash

FILES="/usr/sbin/accept /usr/sbin/pwck /usr/sbin/chroot"

for file in $FILES
do
```

```
  ls -lh $file
done
```

```
$ ./arraytest
lrwxrwxrwx 1 root root 10 Apr 24 11:02 /usr/sbin/accept -> cupsaccept
-rwxr-xr-x 1 root root 54K Mar 22 14:32 /usr/sbin/pwck
-rwxr-xr-x 1 root root 43K Jan 14 07:17 /usr/sbin/chroot
```

Im obigen Beispiel `friendly2.sh` sehen Sie, dass wir mit einer Reihe von Werten arbeiten, die in einer einzigen Variablen `$@` enthalten sind. Der Übersichtlichkeit halber nennen wir die letztgenannte Variable `username`:

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    for username in $@
11    do
12      echo "Hello $username!"
13    done
14    exit 0
15  fi
```

Denken Sie daran, dass die Variable, die Sie hier definieren, beliebig benannt werden kann und dass alle Zeilen in `do… done` einmal für jedes Element des Arrays ausgeführt werden. Betrachten wir die Ausgabe unseres Skripts:

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol!
Hello Dave!
Hello Henry!
```

Nehmen wir nun an, wir wollen unseren Output etwas menschlicher gestalten, indem wir unseren Gruß in eine Zeile setzen.

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    echo -n "Hello $1"
11    shift
12    for username in $@
13    do
14      echo -n ", and $username"
15    done
16    echo "!"
17    exit 0
18  fi
```

Einige Anmerkungen:

- Die Verwendung von `-n` mit `echo` unterdrückt den Zeilenumbruch nach der Ausgabe, d.h. alle Ausgaben werden auf dieselbe Zeile gedruckt, und der Zeilenumbruch folgt erst nach dem `!` in Zeile 16.
- Der Befehl `shift` entfernt das erste Element unseres Arrays. So wird aus:

`0`	`1`	`2`
`Carol`	`Dave`	`Henry`

das Folgende:

`0`	`1`
`Dave`	`Henry`

Betrachten wir die Ausgabe:

```
$ ./friendly2.sh Carol
Hello Carol!
$ ./friendly2.sh Carol Dave Henry
Hello Carol, and Dave, and Henry!
```

Reguläre Ausdrücke bei der Fehlerprüfung

Nehmen wir an, wir wollen alle Argumente, die der Benutzer eingibt, überprüfen, z.B. dass alle an `friendly2.sh` übergebenen Namen *nur Buchstaben* enthalten und alle Sonderzeichen oder Zahlen einen Fehler werfen. Für eine solche Fehlerprüfung nutzen wir `grep`.

Erinnern Sie sich daran, dass wir reguläre Ausdrücke mit `grep` verwenden können.

```
$ echo Animal | grep "^[A-Za-z]*$"
Animal
$ echo $?
0
```

```
$ echo 4n1ml | grep "^[A-Za-z]*$"
$ echo $?
1
```

Das `^` und das `$` kennzeichnen den Anfang und das Ende der Zeile, das `[A-Za-z]` kennzeichnet einen Bereich von Buchstaben, Groß- oder Kleinschreibung. Das `*` ist ein *Quantisierer* und modifiziert unseren Buchstabenbereich von null bis zu (beliebig) vielen Buchstaben. Zusammenfassend lässt sich sagen, dass unser `grep` erfolgreich ist, wenn die Eingabe *ausschließlich* Buchstaben umfasst — andernfalls schlägt es fehl.

Im nächsten Schritt müssen wir dafür sorgen, dass `grep` Exit Codes zurückgibt, abhängig davon, ob es eine Übereinstimmung gab oder nicht. Eine positive Übereinstimmung gibt `0` zurück, andernfalls wird der Wert `1` zurückgegeben. Wir können dies verwenden, um die Argumente in unserem Skript zu testen.

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    for username in $@
11    do
12      echo $username | grep "^[A-Za-z]*$" > /dev/null
13      if [ $? -eq 1 ]
```

```
14      then
15        echo "ERROR: Names must only contains letters."
16        exit 2
17      else
18        echo "Hello $username!"
19      fi
20    done
21    exit 0
22  fi
```

In Zeile 12 leiten wir die Standardausgabe an `/dev/null` um, was eine einfache Möglichkeit ist, sie zu unterdrücken, denn wir wollen keine Ausgabe des Befehls `grep` sehen, sondern nur seinen Exit Code testen, was in Zeile 13 geschieht. Beachten Sie auch, dass wir einen Exit Code `2` verwenden, um ein ungültiges Argument anzuzeigen. Es ist gute Praxis, verschiedene Exit Codes zu verwenden, um verschiedene Fehler anzuzeigen; so kann ein erfahrener Benutzer diese Exit Codes zur Fehlersuche nutzen.

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol!
Hello Dave!
Hello Henry!
$ ./friendly2.sh 42 Carol Dave Henry
ERROR: Names must only contains letters.
$ echo $?
2
```

Geführte Übungen

1. Lesen Sie das folgende `script1.sh`:

```
#!/bin/bash

if [ $# -lt 1 ]
then
  echo "This script requires at least 1 argument."
  exit 1
fi

echo $1 | grep "^[A-Z]*$" > /dev/null
if [ $? -ne 0 ]
then
  echo "no cake for you!"
  exit 2
fi

echo "here's your cake!"
exit 0
```

   Wie lautet die Ausgabe der folgenden Befehle?

   - `./script1.sh`
   - `echo $?`
   - `./script1.sh cake`
   - `echo $?`
   - `./script1.sh CAKE`
   - `echo $?`

2. Lesen Sie das folgende Skript `script2.sh`:

```
for filename in $1/*.txt
do
   cp $filename $filename.bak
done
```

Beschreiben Sie den Zweck des Skripts so, wie Sie es verstehen.

Offene Übungen

1. Erstellen Sie ein Skript, das beliebig viele Argumente vom Benutzer entgegennimmt, und drucken Sie nur solche Argumente, die Zahlen größer als 10 sind.

Zusammenfassung

In diesem Abschnitt haben Sie gelernt:

- Was Exit Codes sind, was sie bedeuten und wie man sie implementiert.
- Wie man den Exit Code eines Befehls überprüft.
- Was for-Schleifen sind, und wie man sie mit Arrays verwendet.
- Wie man `grep`, reguläre Ausdrücke und Exit Codes verwendet, um Benutzereingaben in Skripten zu überprüfen.

Befehle, die in den Übungen verwendet werden:

`shift`
: Entfernt das erste Element eines Arrays.

Spezielle Variablen:

`$?`
: Enthält den Exit Code des zuletzt ausgeführten Befehls.

`$@`, **`$*`**
: Enthält alle Argumente, die an das Skript übergeben werden, als Array.

Lösungen zu den geführten Übungen

1. Lesen Sie das folgende `script1.sh`:

```
#!/bin/bash

if [ $# -lt 1 ]
then
  echo "This script requires at least 1 argument."
  exit 1
fi

echo $1 | grep "^[A-Z]*$" > /dev/null
if [ $? -ne 0 ]
then
  echo "no cake for you!"
  exit 2
fi

echo "here's your cake!"
exit 0
```

   Wie lautet die Ausgabe der folgenden Befehle?

   - Befehl: `./script1.sh`

     Ausgabe: `This script requires at least 1 argument.`

   - Befehl: `echo $?`

     Ausgabe: `1`

   - Befehl: `./script1.sh cake`

     Ausgabe: `no cake for you!`

   - Befehl: `echo $?`

     Ausgabe: `2`

   - Befehl: `./script1.sh CAKE`

     Ausgabe: `here's your cake!`

- Befehl: `echo $?`

  Ausgabe: `0`

2. Lesen Sie das folgende Skript `script2.sh`:

```
for filename in $1/*.txt
do
   cp $filename $filename.bak
done
```

   Beschreiben Sie den Zweck des Skripts so, wie Sie es verstehen.

   Dieses Skript erstellt Sicherungskopien aller Dateien, die mit `.txt` enden, in einem Unterverzeichnis, das im ersten Argument definiert ist.

Lösungen zu den offenen Übungen

1. Erstellen Sie ein Skript, das beliebig viele Argumente vom Benutzer entgegennimmt, und drucken Sie nur solche Argumente, die Zahlen größer als 10 sind.

```
#!/bin/bash

for i in $@
do
  echo $i | grep "^[0-9]*$" > /dev/null
  if [ $? -eq 0 ]
  then
    if [ $i -gt 10 ]
    then
      echo -n "$i "
    fi
  fi
done
echo ""
```

Linux Professional Institute

Thema 4: Das Linux-Betriebssystem

 | |

4.1 Ein Betriebssystem auswählen

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 4.1

Gewichtung

1

Hauptwissensgebiete

- Unterschiede zwischen Windows, OS X und Linux
- Lebenszyklus-Management von Distributionen

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- GUI versus Befehlszeile, Desktop-Konfiguration
- Wartungszyklen, Beta und Stabil

4.1 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	4 Das Linux-Betriebssystem
Lernziel:	4.1 Ein Betriebssystem auswählen
Lektion:	1 von 1

Einführung

Ob Sie Ihr Computersystem zu Hause, an der Universität oder in einem Unternehmen verwenden — Sie müssen sich für ein Betriebssystem entscheiden. Diese Entscheidung können Sie selbst treffen, insbesondere wenn es sich um Ihren Computer handelt, aber Sie können auch für die Auswahl der Systeme in Ihrem Unternehmen verantwortlich sein. Wie so oft helfen Informationen zu den verfügbaren Optionen, eine verantwortungsvolle Entscheidung zu treffen. Mit dieser Lektion geben wir Ihnen für die Auswahl des Betriebssystems wichtige Informationen an die Hand.

Was ist ein Betriebssystem?

Bevor wir mit unserer Erkundungsreise auf der Suche nach dem geeigneten Betriebssystems beginnen, ist zu klären, was wir unter dem Begriff verstehen. Das Betriebssystem ist das Herzstück des Computers und macht es erst möglich, Anwendungen darin bzw. darauf laufen zu lassen. Zudem umfasst das Betriebssystem Treiber für den Zugriff auf die Hardware des Computers, wie Festplatten und Partitionen, Bildschirme, Tastaturen, Netzwerkkarten usw. Oft kürzen wir das Betriebssystem mit *OS* (*Operating System*) ab. Heute gibt es zahlreiche Betriebssysteme für Rechner sowohl in Unternehmen wie auch zu Hause. Um die Auswahl zu

vereinfachen, können wir sie wie folgt gruppieren:

- Linux-basierte Betriebssysteme
  - Enterprise Linux
  - Consumer Linux
- Unix
- macOS
- Windows-basierte Betriebssysteme
  - Windows Server
  - Windows Desktops

Auswahl einer Linux-Distribution

Der Linux-Kernel und Linux-Distributionen

Spricht man von Linux-Distributionen, ist Linux das Betriebssystem. Linux ist der *Kernel*, das Herzstück jeder Linux-Distribution. Die Software des Linux-Kerns wird von einer Gruppe von Einzelpersonen unter der Leitung von Linus Torvalds verwaltet. Torvalds ist bei einem Industriekonsortium namens The Linux Foundation angestellt, um am Linux-Kernel zu arbeiten.

NOTE Der Linux-Kernel wurde 1991 von Linus Torvalds, einem Studenten aus Finnland, entwickelt. Das erste Kernel-Release unter der GNU General Public License Version 2 (GPLv2) war 1992 die Version 0.12.

Linux-Kernel

Wie bereits erwähnt, laufen alle Linux-Distributionen auf dem gleichen Betriebssystem: Linux.

Linux-Distribution

Spricht man von Red Hat Linux oder Ubuntu Linux, bezieht man sich auf die jeweilige *Linux-Distribution*. Eine Linux-Distribution wird mit einem Linux-Kernel und einer Umgebung ausgeliefert, die den Kernel so nutzbar macht, dass wir mit ihm interagieren können. Mindestens benötigen wir eine Kommandozeilen-Shell wie Bash und eine Reihe grundlegender Befehle, die uns den Zugriff auf das System und dessen Verwaltung ermöglichen. Meist verfügt eine Linux-Distribution natürlich auch über eine vollständige Desktop-Umgebung wie Gnome oder KDE.

Obwohl jede Linux-Distribution das Linux-Betriebssystem verwendet, können und werden sich Distributionen in der Version des verwendeten Betriebssystems unterscheiden, d.h. in der *Version*

des Linux-Kernels, die beim Booten der Distribution verwendet wird.

TIP

Wenn Sie Zugriff auf eine Linux-Kommandozeile haben, können Sie die Version des Linux-Kernels, den Sie ausführen, einfach ermitteln, indem Sie das *Kernel Release* anzeigen lassen:

```
$ uname -r
4.15.0-1019-aws
```

Typen von Linux-Distributionen

Es mag naheliegend scheinen, immer die neueste Version des Linux-Kernels auszuführen, aber ganz so einfach ist es nicht. Wir können Linux-Distributionen grob in drei Gruppen einteilen:

- Enterprise Linux-Distributionen
  - Red Hat Enterprise Linux
  - CentOS
  - SUSE Linux Enterprise Server
  - Debian GNU/Linux
  - Ubuntu LTS
- Consumer Linux-Distributionen
  - Fedora
  - Ubuntu non-LTS
  - openSUSE
- Experimentelle and Hacker-Linux-Distributionen
  - Arch
  - Gentoo

Dies ist natürlich nur eine sehr kleine Auswahl von Distributionen; wichtig ist aber der Unterschied zwischen *Enterprise*, *Consumer* und *experimentellen* Distributionen und warum es sie gibt.

Enterprise Linux

Distributionen wie CentOS (*Community Enterprise OS*) sind für den Einsatz in großen Unternehmen mit Enterprise-Hardware konzipiert. Die Bedürfnisse großer Unternehmen unterscheiden sich stark von denen kleiner Firmen oder Heimanwendern. Um die

Verfügbarkeit ihrer Dienste zu gewährleisten, stellen Unternehmen höhere Anforderungen an die Stabilität ihrer Hard- und Software, weshalb Enterprise Linux-Distributionen tendenziell ältere Versionen des Kernels und anderer Software umfassen, die bekanntermaßen zuverlässig funktionieren. Häufig portieren die Distributionen wichtige Updates wie Sicherheitskorrekturen zurück in diese stabilen Versionen. Im Gegenzug bieten Enterprise Linux-Distributionen keine Unterstützung für neueste Consumer-Hardware, sondern ältere Versionen von Softwarepaketen. Wie bei Linux-Distributionen für Privatanwender entscheiden sich Unternehmen jedoch auch hier eher für ausgereifte Hardwarekomponenten und bauen ihre Dienste auf stabilen Softwareversionen auf.

Consumer Linux

Distributionen wie Ubuntu sind eher für kleine Unternehmen oder Heim- und Hobbyanwender gedacht, da diese meist auch neueste Hardware in ihren Systemen einsetzen. Solche Systeme benötigen die neuesten Treiber, um das Beste aus der neuen Hardware herauszuholen, aber die Ausgereiftheit solcher Hardware und Treiber wird den Ansprüchen größerer Unternehmen selten gerecht. Für den Verbrauchermarkt ist der neueste Kernel mit den modernsten Treibern genau das, was benötigt wird, auch wenn vielleicht nicht alles ausreichend getestet wurde. Die neueren Linux-Kernel werden über die neuesten Treiber verfügen, um die neueste Hardware zu unterstützen, die wahrscheinlich im Einsatz ist. Gerade bei der Entwicklung von Linux im Gaming-Markt ist es enorm wichtig, dass diesen Anwendern die neuesten Treiber zur Verfügung stehen.

NOTE

Einige Distributionen wie Ubuntu bieten sowohl Consumer-Versionen mit aktueller Software und relativ kurzem Update-Zeitraum, aber auch Versionen mit sogenanntem Long Term Support (LTS) an, die sich eher für Unternehmensumgebungen eignen.

Experimentelle und Hacker-Distributionen

Distributionen wie Arch Linux oder Gentoo Linux entsprechen dem aktuellen Stand der Technik und enthalten die neuesten Softwareversionen, auch wenn diese noch Fehler und ungetestete Features enthalten. Dafür nutzen diese Distributionen ein Rolling-Release-Modell, das es ihnen ermöglicht, jederzeit Updates zu liefern. Diese Distributionen werden von fortgeschrittenen Anwendern eingesetzt, die immer die neueste Software nutzen wollen und sich bewusst sind, dass es jederzeit zu Fehlfunktionen kommen kann, die aber in solchen Fällen in der Lage sind, ihre Systeme zu reparieren.

Kurz gesagt, wenn Sie Linux als Betriebssystem in Betracht ziehen und Enterprise Hardware auf Ihren Servern oder Desktops verwenden, können Sie entweder Enterprise oder Consumer Linux-Distributionen einsetzen. Wenn Sie Consumer Hardware verwenden und das Beste aus den neuesten Hardware-Innovationen machen wollen, dann benötigen Sie wahrscheinlich eine entsprechende Linux-Distribution, um die Anforderungen der Hardware zu erfüllen.

Einige Linux-Distributionen sind miteinander verwandt, so z.B. Ubuntu, das auf Debian Linux basiert und das gleiche Paketierungssystem (DPKG) verwendet. Fedora, als weiteres Beispiel, ist eine Art Testumfeld für Red Hat Enterprise Linux, wo mögliche Features zukünftiger RHEL-Versionen vor ihrer Verfügbarkeit in der Enterprise Distribution überprüft werden können.

Neben den hier genannten Distributionen gibt es noch viele weitere. Ein Vorteil von Linux als Open-Source-Software ist, dass es viele Menschen so entwickeln können, wie es ihrer Meinung nach aussehen sollte. So gibt es viele hundert Distributionen. Für einen Überblick besuchen Sie die Distro Watch Website. Die Macher der Website listen die Top-100-Downloads von Linux-Distributionen auf, und Sie können vergleichen und sehen, was aktuell beliebt ist.

Linux Support-Lebenszyklus

Wie zu erwarten, haben Enterprise Linux-Distributionen längere Support-Zyklen als Consumer- oder Community-Editionen. So bietet etwa Red Hat Enterprise Linux 10 Jahre Support. Red Hat Enterprise Linux 8 wurde im Mai 2019 eingeführt, so dass Software-Updates und Support bis Mai 2029 verfügbar sein werden.

Consumer-Editionen haben meist nur Community-Support über Foren, und Software-Updates sind oft nur für drei Releases verfügbar. Ubuntu 19.04 lieferte beispielsweise Updates mit dem Release 19.10 und endete im Januar 2020. Ubuntu bietet auch Editionen mit Langzeitunterstützung (*Long Term Support*), also mit 5 Jahren Unterstützung der ursprünglichen Version. Die LTS-Version 18.04 wird folglich bis 2023 mit Software-Updates versorgt. Diese LTS-Versionen machen Ubuntu für Unternehmen zu einer Option mit kommerzieller Unterstützung, die von Canonical (dem Unternehmen hinter der Marke Ubuntu) oder unabhängigen Beratungsunternehmen angeboten wird.

NOTE Die Ubuntu-Distributionen verwenden datumsbasierte Versionsnummern im Format JJ.MM. So wurde beispielsweise die Version 19.04 im April 2019 veröffentlicht.

Linux als Desktop

Linux als Desktop-System kann in Unternehmen, in denen sich der Desktop-Support auf kommerzielle Betriebssystemangebote konzentriert, eine größere Herausforderung darstellen. Aber nicht nur der Support kann sich als schwierig erweisen: Ein Unternehmenskunde hat möglicherweise auch große Investitionen in Software-Lösungen getätigt, die ihn an bestimmte Desktop-Betriebssysteme binden. Vor diesem Hintergrund gibt es zahlreiche Beispiele für die Integration von Linux-Desktops in großen Organisationen durch Unternehmen wie Amazon, das sogar eine eigene Linux-Distribution Amazon Linux 2 hat. Diese wird auf der AWS Cloud-Plattform, aber auch intern für Server und Desktops verwendet.

Der Einsatz von Linux in einem kleineren Unternehmen oder zu Hause wird immer einfacher und kann eine lohnende Erfahrung sein, da die Notwendigkeit einer Lizenzierung entfällt und sich der Blick für die Fülle an freier und quelloffener Software öffnet, die für Linux verfügbar ist. Sie werden zudem feststellen, dass es viele verschiedene Desktop-Umgebungen gibt. Am häufigsten sind Gnome und KDE, aber es gibt noch andere. Die Entscheidung bestimmt der persönliche Geschmack.

Linux auf Servern

Linux als Serverbetriebssystem ist im Unternehmensbereich üblich. Die Server werden von Administratoren betreut, die sich auf Linux spezialisiert haben. Selbst bei Tausenden von Benutzern spielt es für diese keine Rolle, mit welchen Servern sie sich verbinden. Das Serverbetriebssystem ist für sie nicht wichtig, und im Allgemeinen unterscheiden sich Client-Anwendungen nicht zwischen Linux und anderen Betriebssystemen im Backend. Und je mehr Anwendungen in lokalen und remote Clouds virtualisiert oder containerisiert werden, desto weiter rückt das Betriebssystem in den Hintergrund — und das Embedded-Betriebssystem ist wahrscheinlich Linux.

Linux in der Cloud

Eine weitere Möglichkeit, sich mit Linux vertraut zu machen, ist dessen Einsatz in einer der vielen Public-Cloud-Lösungen. Mit einem Account bei einem Cloud-Anbieter lassen sich viele verschiedene Linux-Distributionen schnell und einfach aufsetzen.

Nicht-Linux-Betriebssysteme

So unglaublich es scheinen mag, aber es gibt Betriebssysteme, die nicht auf dem Linux-Kernel basieren. Natürlich gab es im Laufe der Jahre viele und einige sind auf der Strecke geblieben, aber es gibt nach wie vor Optionen sowohl für zu Hause wie auch im Büro.

Unix

Vor Linux gab es Unix. Unix wurde zusammen mit der Hardware verkauft, und noch heute sind mehrere kommerzielle Unixe wie AIX und HP-UX auf dem Markt. Während Linux stark von Unix inspiriert war (und der fehlenden Verfügbarkeit für bestimmte Hardware), basiert die Familie der BSD-Betriebssysteme direkt auf Unix. Heute sind FreeBSD, NetBSD und OpenBSD zusammen mit einigen anderen verwandten BSD-Systemen als freie Software erhältlich.

Unix war in Unternehmen stark vertreten, aber mit dem Erfolg von Linux ging der Rückgang von Unix einher. Mit Linux wuchsen auch die Enterprise-Support-Angebote und Unix verschwand allmählich. Solaris, ursprünglich von Sun entwickelt und dann von Oracle übernommen, ist

kürzlich verschwunden — eines der größeren Unix-Betriebssysteme, das von Telekommunikationsunternehmen verwendet und als *Telco Grade Unix* bezeichnet wurde.

Unix-Betriebssysteme sind:

- AIX
- FreeBSD, NetBSD, OpenBSD
- HP-UX
- Irix
- Solaris

macOS

macOS (früher OS X) von Apple stammt aus dem Jahr 2001. Es basiert auf BSD Unix, nutzt die Bash Kommandozeilen-Shell und ist ein benutzerfreundliches System, wenn Sie Unix- oder Linux-Betriebssysteme gewohnt sind. macOS bietet über die Terminalanwendung Zugriff auf die Kommandozeile. Wenn wir den oben genannten Befehl `uname` hier ausführen, erkennen wir ebenfalls das genutzte Betriebssystem:

```
$ uname -s
Darwin
```

NOTE Wir nutzen in diesem Fall die Option `-s`, um den Namen des Betriebssystems zurückzugeben. Wir haben vorher `-r` verwendet, um die Kernel-Versionsnummer zu erfahren.

Microsoft Windows

Nach wie vor ist die Mehrheit der Desktops und Laptops da draußen Windows-basiert. Das Betriebssystem ist ungemein erfolgreich und dominiert seit Jahren den Desktop-Markt. Obwohl es sich um proprietäre Software handelt und nicht kostenlos ist, ist die Betriebssystemlizenz beim Kauf der Hardware meist enthalten, so dass es die einfachste Wahl ist. Es gibt eine breite Unterstützung für Windows bei Hard- und Softwareanbietern, aber viele Open-Source-Anwendungen sind natürlich auch für Windows verfügbar. Die Zukunft für Windows sieht nicht mehr so rosig aus wie früher. Da jetzt weniger Desktops und Laptops verkauft werden, liegt der Schwerpunkt auf dem Tablet- und Telefonmarkt. Dieser wird von Android und Apple dominiert, und es ist schwer für Microsoft, an Boden zu gewinnen.

Als Serverplattform erlaubt Microsoft seinen Kunden nun die Wahl zwischen einer GUI (*Graphical*

User Interface) und einer reinen Kommandozeilenversion. Die Trennung von GUI und Kommandozeile ist wichtig. Meist wird die GUI älterer Microsoft Server geladen, aber niemand nutzt sie… Betrachten Sie einen Active Directory Domain Controller: Benutzer verwenden ihn permanent, um sich gegenüber der Domäne zu authentifizieren, aber sie wird remote vom Desktop der Administratoren verwaltet, nicht vom Server.

Geführte Übungen

1. Welches Projekt bildet die allen Linux-Distributionen gemeinsame Komponente?

CentOS	
Red Hat	
Ubuntu	
Linux Kernel	
CoreOS	

2. Welches Betriebssystem wird für macOS von Apple verwendet?

OS X	
OSX	
Darwin	
MacOS	

3. Inwiefern unterscheidet sich eine Linux-Distribution vom Linux-Kernel?

Der Kernel ist Teil einer Distribution — die Distribution als Anwendungen, die den Kernel umgeben, um ihn nutzbar zu machen.	
Der Kernel ist die Linux-Distribution	
Alle Distributionen, die den gleichen Kernel verwenden, sind gleich	

4. Welches der folgenden ist eine Desktop-Umgebung unter Linux?

Mint	
Elementary	
Zorin	
Gnome	

5. Welche Komponente eines Betriebssystems erlaubt den Zugriff auf Hardware?

Treiber	
Shell	
Dienst	
Anwendung	

Offene Übungen

1. Ermitteln Sie das aktuelle Kernel-Release Ihres Linux-Systems, wenn Sie Zugriff auf die Befehlszeile haben.

2. Finden Sie mit Ihrer bevorzugten Suchmaschine die für Sie verfügbaren Public-Cloud-Anbieter. Dazu gehören AWS, Google Cloud, Rackspace und viele mehr. Wählen Sie einen aus und finden Sie heraus, welche Betriebssysteme bereitgestellt werden.

Zusammenfassung

In diesem Abschnitt haben Sie gelernt, wie Sie zwischen verschiedenen gängigen Betriebssystemen unterscheiden können:

- Linux-basierte Betriebssysteme
- Unix
- macOS
- Windows-basierte Betriebssysteme

Innerhalb der Linux-Kategorie konnten wir die Auswahl weiter in Distributionen mit langfristigem Support und solche mit kürzerem Support-Zyklus unterteilen. LTS-Versionen, die besser für Unternehmen geeignet sind, und kurzfristige Unterstützung, die sich an Privat- und Hobbyanwender richtet.

- Enterprise Linux-Distributionen
  - Red Hat Enterprise Linux
  - CentOS
  - SUSE Linux Enterprise Server
  - Debian GNU/Linux
  - Ubuntu LTS
- Consumer Linux-Distributionen
  - Fedora
  - Ubuntu non-LTS
  - openSUSE
- Experimentelle and Hacker-Linux-Distributionen
  - Arch
  - Gentoo

Lösungen zu den geführten Übungen

1. Welches Projekt bildet die allen Linux-Distributionen gemeinsame Komponente?

CentOS	
Red Hat	
Ubuntu	
Linux Kernel	X
CoreOS	

2. Welches Betriebssystem wird für macOS von Apple verwendet?

OS X	
OSX	
Darwin	X
MacOS	

3. Inwiefern unterscheidet sich eine Linux-Distribution vom Linux-Kernel?

Der Kernel ist Teil einer Distribution — die Distribution als Anwendungen, die den Kernel umgeben, um ihn nutzbar zu machen.	X
Der Kernel ist die Linux-Distribution	
Alle Distributionen, die den gleichen Kernel verwenden, sind gleich	

4. Welches der folgenden ist eine Desktop-Umgebung unter Linux?

Mint	
Elementary	
Zorin	
Gnome	X

5. Welche Komponente eines Betriebssystems erlaubt den Zugriff auf Hardware?

Treiber	X
Shell	
Dienst	
Anwendung	

Lösungen zu den offenen Übungen

1. Ermitteln Sie das aktuelle Kernel-Release Ihres Linux-Systems, wenn Sie Zugriff auf die Befehlszeile haben.

   ```
   $ uname -r
   4.15.0-47-generic
   ```

2. Finden Sie mit Ihrer bevorzugten Suchmaschine die für Sie verfügbaren Public-Cloud-Anbieter. Dazu gehören AWS, Google Cloud, Rackspace und viele mehr. Wählen Sie einen aus und finden Sie heraus, welche Betriebssysteme bereitgestellt werden.

   AWS ermöglicht es Ihnen beispielsweise, viele Linux-Distributionen wie Debian, Red Hat, SUSE oder Ubuntu sowie Windows einzusetzen.

4.2 Verständnis von Computer-Hardware

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 4.2

Gewichtung

2

Hauptlernziele

- Hardware

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

- Motherboards, Prozessoren, Netzteile, optische Laufwerke, Peripheriegeräte
- Festplatten, SSD und Partitionen, `/dev/sd*`
- Treiber

4.2 Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	4 Das Linux-Betriebssystem
Lernziel:	4.2 Verständnis von Computer-Hardware
Lektion:	1 von 1

Einführung

Ohne Hardware ist Software nichts anderes als eine Form von Literatur. Hardware verarbeitet die von der Software beschriebenen Befehle und stellt Mechanismen für Speicherung sowie Eingabe und Ausgabe bereit. Auch die Cloud läuft letztlich auf Hardware.

Als Betriebssystem ist Linux unter anderem dafür verantwortlich, Software mit Schnittstellen für den Zugriff auf die Hardware eines Systems auszustatten. Die meisten Konfigurationen übersteigen den Rahmen dieser Lektion, aber Benutzer sind häufig mit Aspekten der Leistung, der Kapazität und anderer Faktoren der Systemhardware konfrontiert, da sie die Fähigkeit eines Systems beeinflussen, bestimmte Anwendungen angemessen zu unterstützen. In dieser Lektion geht es um Hardware als eigenständige physische Geräte mit Standardkonnektoren und Schnittstellen. Die Standards sind relativ statisch, aber Form-, Leistungs- und Kapazitätsmerkmale der Hardware entwickeln sich ständig weiter. Unabhängig davon, wie Veränderungen physische Unterschiede verwischen können, gelten die in dieser Lektion beschriebenen Hardware-Konzepte weiterhin.

NOTE An verschiedenen Stellen in dieser Lektion wird anhand von Befehlszeilenbeispielen der Zugriff auf Hardware-Informationen gezeigt. Die

> meisten Beispiele stammen von einem Raspberry Pi B+, sollten aber für die meisten Systeme gelten. Sie müssen diese Befehle nicht im Einzelnen verstehen, um die Ausführungen nachzuvollziehen.

Netzteile

Alle aktiven Komponenten in einem Computersystem benötigen zum Betrieb Strom. Die meisten Stromquellen sind dafür aber leider nicht geeignet: Computer-Hardware benötigt spezifische Spannungen mit relativ engen Toleranzen, was nicht das ist, was Ihre Steckdose liefert.

Netzteile normalisieren verfügbare Energiequellen. Standardisierte Spannungsanforderungen ermöglichen es Herstellern, Hardwarekomponenten zu entwickeln, die in Systemen überall auf der Welt verwendet werden können. Desktop-Netzteile nutzen üblicherweise Strom aus Steckdosen als Energiequelle. Server-Netzteile sind in der Regel kritischer, so dass sie oft an mehrere Quellen angeschlossen sind, um sicherzustellen, dass sie bei Ausfall einer Quelle weiter funktionieren.

Bei der Nutzung von Strom entsteht Wärme. Übermäßige Hitze kann dazu führen, dass Systemkomponenten langsam arbeiten oder sogar ausfallen. Die meisten Systeme haben darum eine Art Ventilator, der die Luft für eine effizientere Kühlung bewegt. Komponenten wie etwa Prozessoren erzeugen oft Wärme, die allein durch den Luftstrom nicht abgeführt werden kann. Solch heiße Komponenten haben spezielle Rippen, sogenannte Kühlkörper, um die erzeugte Wärme abzuführen. Manche Kühlkörper haben wiederum einen eigenen Ventilator, der für ausreichenden Luftstrom sorgt.

Hauptplatine (Motherboard)

Sämtliche Hardware eines Systems muss miteinander verbunden sein. Die *Hauptplatine* (auch *Motherboard* oder *Mainboard* genannt) normiert diese Verbindung mit standardisierten Steckverbindungen und Formfaktoren. Sie unterstützt zudem die Konfiguration und kümmert sich um die elektrischen Anforderungen dieser Steckverbindungen.

Es gibt eine Vielzahl von Mainboard-Konfigurationen, die verschiedene Prozessoren und Speichersysteme unterstützen, verschiedene Kombinationen standardisierter Steckverbindungen aufweisen und sich an die unterschiedlichen Gehäusegrößen anpassen. Vielleicht mit Ausnahme der Anschlussmöglichkeiten für bestimmte externe Geräte ist die Mainboard-Konfiguration für den Benutzer sehr transparent. Administratoren haben mit der Mainboard-Konfiguration meist dann zu tun, wenn es darum geht, bestimmte Geräte zu identifizieren.

Wird die Stromversorgung eingeschaltet, muss die mainboardspezifische Hardware konfiguriert und initialisiert werden, bevor das System laufen kann. Motherboards verwenden Programme in

einem nichtflüchtigen Speicher, die als *Firmware* bezeichnet werden, um die motherboardspezifische Hardware zu steuern. Die ursprüngliche Form der Motherboard-Firmware wurde als BIOS (*Basic Input/Output System*) bezeichnet, und über die grundlegenden Konfigurationseinstellungen hinaus war BIOS hauptsächlich für die Identifizierung, das Laden und die Übertragung des Betriebs auf ein Betriebssystem wie Linux verantwortlich. Im Laufe der Hardwareentwicklung wurde die Firmware erweitert, um größere Festplatten, Diagnosen, grafische Oberflächen, Netzwerke und andere erweiterte Funktionen unabhängig von jedem geladenen Betriebssystem zu unterstützen. Frühe Versuche, die Firmware über das grundlegende BIOS hinaus zu verbessern, waren oft spezifisch für einen Mainboard-Hersteller. Intel hat einen Standard für erweiterte Firmware definiert, der als EFI (*Extensible Firmware Interface*) bekannt ist. Intel übertrug EFI einer Standardisierungsorganisation, um UEFI (*Unified Extensible Firmware Interface*) zu schaffen. Heute verwenden die meisten Motherboards UEFI. BIOS und EFI sind auf neueren Systemen fast nie zu sehen. Unabhängig davon bezeichnen viele die Firmware des Motherboards immer noch als BIOS.

Es gibt nur sehr wenige Firmware-Einstellungen, die für normale Benutzer von Interesse sind, so dass nur Personen, die für die Konfiguration der Systemhardware verantwortlich sind, sich typischerweise mit der Firmware und ihren Einstellungen auseinandersetzen. Eine der wenigen gemeinhin geänderten Optionen ist die Aktivierung von Virtualisierungserweiterungen moderner CPUs.

Systemspeicher

Der Systemspeicher enthält die Daten und den Programmcode der aktuell laufenden Anwendungen. Spricht man von “Computerspeicher”, ist meist dieser Systemspeicher gemeint. Ein weiterer gebräuchlicher Begriff für den Systemspeicher ist das Akronym RAM (*Random Access Memory*) oder eine Variation dieses Akronyms. Manchmal werden auch Referenzen auf die Form oder Anordnung des Systemspeichers wie DIMM, SIMM oder DDR verwendet.

Physisch gesehen wird der Systemspeicher in der Regel auf einzelnen Platinenmodulen aufgebracht, die in das Motherboard eingesteckt werden. Einzelne Speichermodule sind derzeit in einer Größe von 2 bis 64 GB erhältlich. 4 GB ist für die meisten Standardanwendungen der minimale Systemspeicher, mit dem man planen sollte. 16 GB sind für einzelne Workstations typischerweise mehr als ausreichend, aber auch 16 GB können für Benutzer, die Spiele, Videos oder High-End-Audioanwendungen ausführen, zu wenig sein. Server benötigen häufig 128 oder sogar 256 GB Speicher, um die Benutzerlast effizient zu unterstützen.

In den meisten Fällen erlaubt Linux den Benutzern, den Systemspeicher als Black Box zu behandeln. Eine Anwendung wird gestartet, und Linux kümmert sich um die Zuweisung des benötigten Systemspeichers. Linux gibt den Speicher für andere Anwendungen frei, wenn eine Anwendung abgeschlossen ist. Aber was geschieht, wenn eine Anwendung mehr als den

verfügbaren Systemspeicher benötigt? In diesem Fall verschiebt Linux ungenutzte Anwendungen aus dem Systemspeicher in einen speziellen Plattenbereich, den *Swap Space*, und ungenutzte Anwendungen aus dem Swap Space zurück in den Systemspeicher, wenn sie ausgeführt werden müssen.

Systeme ohne dedizierte Video-Hardware nutzen häufig einen Teil des Systemspeichers (oft 1 GB) als Videoanzeigespeicher, was den effektiven Systemspeicher reduziert. Dedizierte Video-Hardware hat typischerweise einen eigenen separaten Speicher, der nicht als Systemspeicher verfügbar ist.

Es gibt mehrere Möglichkeiten, Informationen über den Systemspeicher zu erhalten. Für einen Benutzer ist üblicherweise die Gesamtmenge des verfügbaren und des verwendeten Speichers von Interesse. Eine Informationsquelle ist die Ausführung des Befehls `free` mit dem Parameter `-m` zur Ausgabe der Werte in Megabytes:

```
$ free -m
               total        used        free      shared  buff/cache   available
Mem:             748          37          51          14         660         645
Swap:             99           0          99
```

Die erste Zeile gibt den zur Verfügung stehenden Systemgesamtspeicher (`total`), den verwendeten Speicher (`used`) und den freien Speicher (`free`) an. Die zweite Zeile liefert diese Informationen für den Swap Space. Der als `shared` und `buff/cache` angegebene Speicher wird derzeit für andere Systemfunktionen verwendet, obwohl die unter `available` angegebene Menge für Anwendungen genutzt werden könnte.

Prozessoren

Das Wort “Prozessor” impliziert, dass etwas verarbeitet wird. In Computern geht es hauptsächlich um die Verarbeitung elektrischer Signale, die typischerweise so behandelt werden, dass sie einen der Binärwerte 1 oder 0 haben.

Häufig werden der Begriff “Prozessor” und die Abkürzung CPU (*Central Processing Unit*) synonym gebraucht, was technisch nicht korrekt ist. Jeder handelsübliche Rechner hat eine CPU, die die von der Software vorgegebenen binären Befehle verarbeitet. Darum liegt es nahe, Prozessor und CPU gleichzusetzen. Aber moderne Computer haben neben einer CPU oft auch weitere, aufgabenspezifische Prozessoren, etwa eine GPU (*Graphical Processing Unit*). Eine CPU ist also ein Prozessor, aber nicht alle Prozessoren sind CPUs.

Für viele ist die CPU-Architektur ein Hinweis auf die Anweisungen, die der Prozessor unterstützt. Obwohl Intel und AMD Prozessoren herstellen, die dieselben Anweisungen unterstützen, ist es

sinnvoll, nach Anbietern zu unterscheiden, da sie sich herstellerseitig in Bauart, Leistung und Stromverbrauch unterscheiden. Software-Distributionen verwenden diese Bezeichnungen häufig, um den Mindestbedarf an Anweisungen anzugeben, den sie für den Betrieb benötigen:

i386

Verweist auf den 32-Bit-Befehlssatz, der dem Intel 80386 zugeordnet ist.

x86

Verweist typischerweise auf die 32-Bit-Befehlssätze der 80386-Nachfolger, zum Beispiel 80486, 80586 und Pentium.

x64 / x86-64

Referenzprozessoren, die sowohl die 32-Bit- als auch die 64-Bit-Anweisungen der x86-Familie unterstützen.

AMD

Verweist auf die x86-Unterstützung durch AMD-Prozessoren.

AMD64

Verweist auf die x64-Unterstützung durch AMD-Prozessoren.

ARM

Verweist auf eine *Reduced Instruction Set Computer* (RISC) CPU, die nicht auf dem x86-Befehlssatz basiert und häufig von embedded, mobilen, Tablet- und batteriebetriebenen Geräten verwendet wird. Eine Version von Linux für ARM wird vom Raspberry Pi verwendet.

Die Datei `/proc/cpuinfo` enthält detaillierte Informationen über den oder die Prozessoren eines Systems. Leider sind diese Details nicht allgemeinverständlich. Ein übersichtlicheres Ergebnis liefert der Befehl `lscpu`. Hier die Ausgabe von einem Raspberry Pi B+:

```
$ lscpu
Architecture:          armv7l
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
Model:                 4
Model name:            ARMv7 Processor rev 4 (v7l)
CPU max MHz:           1400.0000
```

```
CPU min MHz:           600.0000
BogoMIPS:              38.40
Flags:                 half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32
lpae evtstrm crc32
```

Die ungeheure Vielzahl von Anbietern, Prozessorfamilien und Spezifikationen ist für die meisten Anwender verwirrend, aber es gibt bei CPUs und Prozessoren einige Größen, die Benutzer und Administratoren häufig berücksichtigen müssen, wenn es um die Einrichtung von Betriebsumgebungen geht:

Bitgröße

Bei CPUs bezieht sich diese Zahl sowohl auf die native Größe der von ihr verarbeiteten Daten als auch auf die Menge an Speicher, auf die sie zugreifen kann. Die meisten modernen Systeme sind entweder 32-Bit oder 64-Bit. Wenn eine Anwendung Zugriff auf mehr als 4 Gigabyte Speicher benötigt, muss sie auf einem 64-Bit-System laufen, da 4 Gigabyte den maximalen Adressbereich darstellen, die mit 32 Bit großen Adressen abgebildet werden können. Während 32-Bit-Anwendungen typischerweise auf 64-Bit-Systemen ausgeführt werden können, gilt das umgekehrt nicht.

Taktfrequenz

Oft in Megahertz (MHz) oder Gigahertz (GHz) angegeben, sagt sie aus, wie schnell ein Prozessor Anweisungen verarbeitet. Aber die Prozessorgeschwindigkeit ist nur einer der Faktoren, die Systemreaktionszeiten, Wartezeiten und den Durchsatz beeinflussen. Selbst ein aktiver Multitasking-Anwender nutzt selten mehr als 2 oder 3 Prozent der CPU-Kapazitäten eines gewöhnlichen Desktop-PCs. Nutzen Sie jedoch häufig rechenintensive Anwendungen etwa für Verschlüsselung oder Video-Rendering, so kann die CPU-Geschwindigkeit erheblichen Einfluss auf Durchsatz und Wartezeit haben.

Cache

CPUs benötigen zum Betrieb einen konstanten Strom von Anweisungen und Daten. Die Kosten und der Stromverbrauch eines Multi-Gigabyte-Systemspeichers, auf den mit CPU-Taktgeschwindigkeiten zugegriffen werden kann, wären unerschwinglich. Der CPU-Cache-Speicher ist auf dem CPU-Chip integriert, um einen schnellen Puffer zwischen CPUs und Systemspeicher bereitzustellen. Der Cache ist in mehrere Schichten unterteilt, die üblicherweise als L1, L2, L3 und sogar L4 bezeichnet werden. In Fall von Cache gilt: Je mehr, desto besser.

Kerne (Cores)

Core bezieht sich auf eine einzelne CPU. Zusätzlich zum Core, der eine physische CPU darstellt, ermöglicht *Hyper-Threading Technology* (HTT) einer einzelnen physischen CPU, mehrere

Anweisungen gleichzeitig zu verarbeiten, so dass sie praktisch als mehrere physische CPUs fungiert. Typischerweise werden mehrere physische Kerne als ein einziger physischer Prozessorchip verbaut. Es gibt jedoch Motherboards, die mehrere physische Prozessorchips unterstützen. In der Theorie klingt mehr Kerne zur Verarbeitung von Aufgaben nach besserem Systemdurchsatz. Leider aber lasten Desktop-Anwendungen die CPUs oft nur zu 2 oder 3 Prozent aus, so dass das Hinzufügen weiterer, meist ungenutzter CPUs wahrscheinlich nur zu einer minimalen Verbesserung des Durchsatzes führen wird. Weitere Kerne eignen sich am besten für die Ausführung von Anwendungen, die so geschrieben sind, dass sie mehrere unabhängige Funktionsabläufe aufweisen, wie z.B. Video-Frame-Rendering, Webseiten-Rendering oder VM-Umgebungen mit mehreren Benutzern.

Speicher

Speichergeräte dienen der Aufbewahrung von Programmen und Daten. *Hard Disk Drives* (HDDs) und *Solid State Drives* (SSDs) sind die häufigste Form von Speichergeräten in Servern und Desktops. USB-Speichersticks und optische Geräte wie DVDs werden ebenfalls, aber selten als primäre Speichermedien verwendet.

Wie der Name schon sagt, speichert ein Festplattenlaufwerk Informationen auf einer oder mehreren starren physischen Platten, die mit magnetischen Materialien bedeckt sind, um die Speicherung zu ermöglichen. Die Platten befinden sich in einem abgedichteten Gehäuse, da Staub, kleine Partikel und sogar Fingerabdrücke die Fähigkeit der HDD beeinträchtigen würden, die magnetischen Medien zu lesen und zu beschreiben.

SSDs sind deutlich anspruchsvollere Varianten von USB-Sticks mit wesentlich größerer Kapazität. Sie speichern Informationen in Mikrochips, so dass es keine beweglichen Bauteile gibt.

Obwohl die zugrunde liegenden Technologien für HDDs und SSDs unterschiedlich sind, gibt es wichtige Vergleichsparameter: Die Kapazität von Festplatten basiert auf der Skalierung physischer Komponenten, während die SSD-Kapazität von der Anzahl der Mikrochips abhängt. Pro Gigabyte kosten SSDs zwischen dem Drei- und Zehnfachen einer Festplatte. Zum Lesen oder Schreiben muss sich eine bestimmte Stelle einer Festplatte an einen bestimmten Ort drehen, während SSDs Direktzugriff erlauben. Die Zugriffsgeschwindigkeiten auf SSDs sind in der Regel 3 bis 5 mal höher als bei HDDs, und da sie keine beweglichen Teile haben, verbrauchen SSDs weniger Strom und sind zuverlässiger als Festplatten.

Die Speicherkapazität von Festplatten und SSDs nimmt ständig zu. Festplatten mit 5 und SSDs mit 1 Terabyte sind heute üblich. Dennoch ist hohe Speicherkapazität nicht immer von Vorteil: Fällt ein Speichermedium aus, stehen sämtliche darauf enthaltenen Daten nicht mehr zur Verfügung, und natürlich dauert die Sicherung länger, wenn mehr Informationen zu sichern sind. Bei Anwendungen, die viele Daten lesen und schreiben, können Latenz und Leistung wichtiger sein

als Kapazität.

Moderne Systeme verwenden SCSI (*Small Computer System Interface*) oder SATA (*Serial AT Attachment*) für die Verbindung mit Speichermedien. Diese Interfaces werden typischerweise durch entsprechende Stecker auf der Hauptplatine unterstützt. Die Erstladung erfolgt von einem an die Hauptplatine angeschlossenen Speichermedium. Die Firmware-Einstellungen definieren die Reihenfolge, in der bei diesem ersten Laden auf die Geräte zugegriffen wird.

Speichersysteme, die als RAID (*Redundant Array of Independent Disks*) bezeichnet werden, sind eine gängige Implementierung zur Vermeidung von Datenverlust. Ein RAID-Array besteht aus mehreren physischen Geräten, die doppelte Kopien von Informationen enthalten. Fällt ein Gerät aus, sind alle Informationen weiterhin verfügbar. Verschiedene physische RAID-Konfigurationen werden als 0, 1, 5, 6 und 10 bezeichnet. Jede Bezeichnung steht für eine bestimmte Speichergröße, Leistungsmerkmale und Möglichkeiten, redundante Daten oder Prüfsummen zur Datenwiederherstellung zu speichern. Abgesehen von etwas Konfigurationsaufwand ist der Einsatz von RAID für die Benutzer weitgehend transparent.

Speichergeräte lesen und schreiben üblicherweise Daten in Blöcken von Bytes. Mit dem Befehl `lsblk` können Sie die einem System zur Verfügung stehenden Blockdevices auflisten. Das folgende Beispiel stammt von einem Raspberry Pi mit einer SD-Karte als Speichermedium. Die Details der Ausgabe werden in den folgenden Abschnitten behandelt:

```
$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0    0 29.7G  0 disk
+-mmcblk0p1 179:1    0 43.9M  0 part /boot
+-mmcblk0p2 179:2    0 29.7G  0 part /
```

Partitionen

Ein Speichergerät ist praktisch eine lange Folge von Speicherplätzen. Partitionierung ist der Mechanismus, der Linux sagt, ob es diese Speicherorte als eine einzige Sequenz oder mehrere unabhängige Sequenzen sehen soll. Jede Partition wird wie ein einzelnes Gerät behandelt. Die meisten Partitionen werden bei der ersten Konfiguration eines Systems erstellt. Wenn Änderungen erforderlich sind, stehen administrative Tools zur Verfügung, um die Gerätepartitionierung zu verwalten.

Warum also sind mehrere Partitionen wünschenswert? Einige Beispiele für die Verwendung von Partitionen sind die Verwaltung des verfügbaren Speichers, die Isolierung von Verschlüsselungs-Overhead oder die Unterstützung mehrerer Dateisysteme. Partitionen ermöglichen es, ein

einziges Speichergerät zu haben, das unter verschiedenen Betriebssystemen booten kann.

Zwar erkennt Linux die Speichersequenz eines Raw-Device, aber ein Raw-Device kann nicht “einfach so” verwendet werden. Um ein Raw-Device zu nutzen, muss es formatiert werden. Das Formatieren schreibt ein Dateisystem auf ein Gerät und bereitet es für Dateioperationen vor. Ohne ein Dateisystem kann ein Gerät nicht für dateibezogene Operationen genutzt werden.

Benutzer sehen Partitionen so, als seien sie einzelne Geräte. So übersieht man leicht, dass es sich um ein einzelnes physisches Gerät handelt. Insbesondere Device-zu-Device-Operationen, die tatsächlich Partition-zu-Partition-Operationen sind, haben nicht die erwartete Performance. Ein einzelnes Gerät ist ein physischer Mechanismus mit einem Satz Lese-/Schreib-Hardware. Was noch wichtiger ist: Sie können die Partitionen eines einzelnen physischen Geräts nicht für ein fehlertolerantes Design verwenden. Wenn das Gerät ausfällt, fallen alle Partitionen aus, so dass es keine Fehlertoleranz gäbe.

NOTE *Logical Volume Manager* (LVM) ist eine Softwarefunktion, die es Administratoren ermöglicht, einzelne Festplatten und Festplattenpartitionen zu kombinieren und so zu behandeln, als wären sie ein einzelnes Laufwerk.

Peripheriegeräte

Server und Workstations benötigen für den Betrieb eine Kombination aus CPU, Systemspeicher und Speicher. Aber diese grundlegenden Komponenten interagieren nicht direkt mit der Außenwelt. Peripheriegeräte sind die Geräte, die den Systemen Input, Output und Zugriff auf den Rest der realen Welt ermöglichen.

Die meisten Motherboards verfügen über eingebaute externe Anschlüsse und Firmware-Unterstützung für gängige ältere Peripherieschnittstellen, die Geräte wie Tastatur, Maus, Sound, Video und Netzwerk unterstützen. Neuere Motherboards verfügen in der Regel über einen Ethernet-Anschluss zur Unterstützung von Netzwerken, einen HDMI-Anschluss zur Unterstützung grundlegender grafischer Anforderungen und einen oder mehrere USB-Anschlüsse (*Universal Serial Bus*) für die meisten anderen Alltagsgeräte. Es gibt mehrere Versionen von USB mit unterschiedlichen Geschwindigkeiten und physikalischen Eigenschaften. Mehrere Versionen von USB-Ports sind auf einem einzigen Motherboard üblich.

Motherboards können auch einen oder mehrere Erweiterungssteckplätze haben, die es dem Benutzer ermöglichen, spezielle Leiterplatten, so genannte Erweiterungskarten, hinzuzufügen, etwa für benutzerdefinierte, ältere und nicht standardmäßige Peripheriegeräte. Grafik-, Sound- und Netzwerkschnittstellen sind gängige Erweiterungskarten. Erweiterungskarten unterstützen auch RAID und Legacy-Schnittstellen im Sonderformat mit seriellen und parallelen Verbindungen.

System on a Chip (SoC)-Konfigurationen bieten Leistungs-, Performance-, Platz- und Zuverlässigkeitsvorteile gegenüber Mainboard-Konfigurationen, indem sie Prozessoren, Systemspeicher, SSD und Hardware zur Steuerung von Peripheriegeräten als ein einziges integriertes Schaltungspaket zusammenfassen. Die von SoC-Konfigurationen unterstützten Peripheriegeräte sind durch die gepackten Komponenten begrenzt, so dass SoC-Konfigurationen in der Regel für bestimmte Einsatzzwecke entwickelt werden, zum Beispiel Telefone, Tablets und andere mobile Geräte.

Einige Systeme enthalten Peripheriegeräte. Laptops ähneln Workstations, verfügen aber über standardmäßige Anzeige-, Tastatur- und Mausperipheriegeräte. All-in-One-Systeme ähneln Laptops, erfordern aber Maus- und Tastaturperipheriegeräte. Motherboard oder SoC-basierte Controller werden oft mit integrierten Peripheriegeräten geliefert, die für einen bestimmten Zweck konzipiert sind.

Treiber und Gerätedateien

In dieser Lektion ging es bislang um Informationen zu Prozessoren, Speicher, Festplatten, Partitionen, Formatierung und Peripheriegeräten. Wenn sich aber Benutzer mit Details zu jedem dieser Geräte in ihrem System befassen müssten, wären diese Systeme unbrauchbar. Ebenso müssten Softwareentwickler ihren Code für jedes neue oder veränderte Gerät, das sie unterstützen, ändern.

Die Lösung für diese Detail-Probleme bieten Gerätetreiber (*Device Driver* oder kurz *Driver*). Treiber akzeptieren einen Standardsatz von Anfragen und übersetzen diese dann in die entsprechenden Steuerungsaktivitäten des Geräts. Gerätetreiber machen es Ihnen und den Anwendungen, die Sie ausführen, möglich, die Datei `/home/carol/stuff` zu lesen, ohne sich Gedanken darüber zu machen, ob sich diese Datei auf einer Festplatte, einem Solid State Drive, einem Memory Stick, einem verschlüsselten Speicher oder einem anderen Gerät befindet.

Gerätedateien liegen im Verzeichnis `/dev` und identifizieren physische Geräte, Gerätezugriff und unterstützte Treiber. In modernen Systemen, die SCSI- oder SATA-basierte Speichergeräte verwenden, beginnt der Dateiname der Spezifikation üblicherweise mit dem Präfix `sd`, gefolgt von einem Buchstaben wie `a` oder `b`, der auf ein physisches Gerät verweist. Nach dem Präfix und der Gerätekennung kommt eine Nummer, die eine Partition in dem physischen Gerät angibt. `/dev/sda` würde also auf das gesamte erste Speichermedium verweisen, während `/dev/sda3` die Partition 3 im ersten Speicher bezeichnet. Die Gerätedatei für jeden Gerätetyp weist eine dem Gerät entsprechende Namenskonvention auf. Die Erläuterung aller möglichen Namenskonventionen geht weit über den Rahmen dieser Lektion hinaus, aber es ist wichtig zu wissen, dass diese Konventionen Systemadministration überhaupt erst möglich machen.

Statt den gesamten Inhalt des Verzeichnisses `/dev` zu betrachten, schauen wir uns den Eintrag für

ein Speichermedium an: Gerätedateien für SD-Karten verwenden typischerweise das Präfix `mmcblk`:

```
$ ls -l mmcblk*
brw-rw---- 1 root disk 179, 0 Jun 30 01:17 mmcblk0
brw-rw---- 1 root disk 179, 1 Jun 30 01:17 mmcblk0p1
brw-rw---- 1 root disk 179, 2 Jun 30 01:17 mmcblk0p2
```

Das Listing der Details einer Gerätedatei unterscheidet sich von dem normaler Dateien:

- Im Gegensatz zu einer Datei oder einem Verzeichnis ist der erste Buchstabe des Berechtigungsfeldes `b`. Das zeigt an, dass Daten nicht in einzelnen Zeichen, sondern in Blöcken vom Gerät gelesen und auf das Gerät geschrieben werden.
- Das Größenfeld besteht aus zwei, durch Komma getrennten Werten, nicht aus einem einzelnen Wert. Der erste Wert gibt im Allgemeinen einen bestimmten Treiber innerhalb des Kernels an und der zweite Wert ein bestimmtes Gerät, das der Treiber steuert.
- Der Dateiname verwendet eine Nummer für das physische Gerät, so dass sich die Namenskonvention anpasst, indem das Partitionssuffix als `p` gefolgt von einer Ziffer angegeben wird.

NOTE Jedes Device sollte einen Eintrag in `/dev` haben. Da der Inhalt des Verzechnisses `/dev` bei der Installation erstellt wird, gibt es oft Einträge für alle möglichen Treiber und Geräte, auch wenn kein physisches Gerät vorhanden ist.

Geführte Übungen

1. Erläutern Sie die folgenden Begriffe:

Prozessor	
CPU	
GPU	

2. Wenn Sie vor allem Anwendungen zur (sehr rechenintensiven) Videobearbeitung ausführen, welche Komponenten und Eigenschaften haben Ihrer Meinung nach den größten Einfluss auf die Benutzbarkeit des Systems:

CPU-Kerne	
CPU-Geschwindigkeit	
Verfügbarer Systemspeicher	
Speicher	
GPU	
Videoanzeige	
Keines der genannten	

3. Wie würde Ihrer Meinung nach der Name der Gerätedatei in `/dev` für die Partition 3 des dritten SATA-Laufwerks in einem System lauten?

`sd3p3`	
`sdcp3`	
`sdc3`	
None of the above	

Offene Übungen

1. Führen Sie den Befehl `lsblk` auf Ihrem System aus. Identifizieren Sie die unten genannten Punkte. Wenn Ihnen kein System zur Verfügung steht, nehmen Sie die Ausgabe von `lsblk -f` auf dem Raspberry Pi aus dieser Lektion im Abschnitt “Speicher”:

```
$ lsblk -f
NAME        FSTYPE LABEL  UUID                                 MOUNTPOINT
mmcblk0
+-mmcblk0p1 vfat   boot   9304-D9FD                            /boot
+-mmcblk0p2 ext4   rootfs 29075e46-f0d4-44e2-a9e7-55ac02d6e6cc /
```

   - Art und Anzahl der Geräte
   - Partitionsstruktur jedes Geräts
   - Dateisystemtyp und Einhängepunkt (Mountpoint) für jede Partition

Zusammenfassung

Ein System ist die Summe seiner Komponenten. Verschiedene Komponenten beeinflussen Kosten, Leistung und Benutzbarkeit auf unterschiedliche Weise. Es gibt zwar allgemeine Konfigurationen für Workstations und Server, aber es gibt nicht die eine optimale Konfiguration.

Lösungen zu den geführten Übungen

1. Erläutern Sie die folgenden Begriffe:

   Prozessor

   Ein allgemeiner Begriff für jede Art von Recheneinheit, der oft falsch als Synonym für CPU verwendet wird.

   CPU

   Die Central Processing Unit (zentrale Verarbeitungseinheit), die allgemeine Berechnungsaufgaben unterstützt.

   GPU

   Die Graphical Processing Unit (grafische Verarbeitungseinheit), die für Berechnungen zur Darstellung von Grafik optimiert ist.

2. Wenn Sie vor allem Anwendungen zur (sehr rechenintensiven) Videobearbeitung ausführen, welche Komponenten und Eigenschaften haben Ihrer Meinung nach den größten Einfluss auf die Benutzbarkeit des Systems:

   CPU-Kerne

   Ja. Mehrere Kerne unterstützen die für die Videobearbeitung erforderlichen gleichzeitigen Präsentations- und Rendering-Aufgaben.

   CPU-Geschwindigkeit

   Ja. Die Videowiedergabe erfordert eine erhebliche Rechenleistung.

   Verfügbarer Systemspeicher

   Wahrscheinlich. Das unkomprimierte Video, das bei der Bearbeitung verwendet wird, ist groß. Mehrzwecksysteme verfügen oft über 8 Gigabyte Speicher. 16 oder sogar 32 Gigabyte Speicherplatz ermöglichen es dem System, mehr Frames mit unkomprimiertem Video zu verarbeiten, was die Bearbeitung effizienter macht.

   Speicher

   Ja. Videodateien sind groß. Der Overhead von lokalen SSD-Laufwerken unterstützt eine effizientere Übertragung. Langsamere Netzlaufwerke sind wahrscheinlich kontraproduktiv.

   GPU

   Nein. Die GPU beeinflusst in erster Linie die Darstellung des gerenderten Videos.

Videoanzeige

Nein. Die Videoanzeige wirkt sich in erster Linie auf die Darstellung des gerenderten Videos aus.

Keines der genannten

Nein. Einige dieser Faktoren haben offensichtliche Auswirkungen darauf, wie nutzbar Ihr System wäre.

3. Wie würde Ihrer Meinung nach der Name der Gerätedatei in `/dev` für die Partition 3 des dritten SATA-Laufwerks in einem System lauten?

`sd3p3`	Not correct. Drive 3 would be `sdc` not `sd3`
`sdcp3`	Not correct. Partition 3 would be `3` not `p3`
`sdc3`	Correct
None of the above	Not correct. The correct answer is one of the choices.

Lösungen zu den offenen Übungen

1. Führen Sie den Befehl `lsblk` auf Ihrem System aus. Identifizieren Sie die unten genannten Punkte. Wenn Ihnen kein System zur Verfügung steht, nehmen Sie die Ausgabe von `lsblk -f` auf dem Raspberry Pi aus dieser Lektion im Abschnitt “Speicher”:

```
$ lsblk -f
NAME        FSTYPE LABEL  UUID                                 MOUNTPOINT
mmcblk0
+-mmcblk0p1 vfat   boot   9304-D9FD                            /boot
+-mmcblk0p2 ext4   rootfs 29075e46-f0d4-44e2-a9e7-55ac02d6e6cc /
```

   Die folgenden Antworten basieren auf dem oben genannten Beispiel. Ihre Antworten können anders ausfallen:

   Art und Anzahl der Geräte

   Es gibt nur ein Gerät: `mmcblk0`. Laut Konvention handelt es sich bei `mmcblk` um eine SD-Speicherkarte.

   Partitionsstruktur jedes Geräts

   Es gibt zwei Partitionen: `mmcblk0p1` und `mmcblk0p2`.

   Dateisystemtyp und Einhängepunkt (Mountpoint) für jede Partition

   Partition 1 verwendet das Dateisystem `vfat`. Es wird zum Booten des Systems verwendet und als `/boot` eingebunden. Partition 2 verwendet das Dateisystem `ext4`. Es wird als primäres Dateisystem verwendet und als `/` eingehängt.

4.3 Wo Daten gespeichert werden

Referenz zu den LPI-Lernzielen

Linux Essentials version 1.6, Exam 010, Objective 4.3

Gewichtung

3

Hauptwissensgebiete

- Programme und Konfiguration
- Prozesse
- Speicheradressen
- Systembenachrichtigungen
- Protokollierung

Auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme

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

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` 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.

Lösungen zu den geführten 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`	`which swapon`	`/sbin/swapon`	Ja
`kill`	`which kill`	`/bin/kill`	Nein
`cut`	`which cut`	`/usr/bin/cut`	Nein
`usermod`	`which usermod`	`/usr/sbin/usermod`	Ja
`cron`	`which cron`	`/usr/sbin/cron`	Ja
`ps`	`which ps`	`/bin/ps`	Nein

2. Wo sind die folgenden Dateien zu finden?

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

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

Zahlenelement	**Bedeutung**
`4`	Kernel-Version
`15`	Major-Revision
`0`	Minor-Revision
`50`	Patch-Nummer

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

   `ls /dev/sd*`

Lösungen zu den offenen 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?

     `/dev/hd*`

   - Und moderne NVMe (Non-Volatile Memory Express) Laufwerke?

     `/dev/nvme*`

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`	`MemTotal` / `SwapTotal`
`free`	`MemFree` / `SwapFree`
`shared`	`Shmem`
`buff/cache`	`Buffers`, `Cached` und `SReclaimable`
`available`	`MemAvailable`

4.3 Lektion 2

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

Einführung

Nachdem wir uns Programme und deren Konfigurationsdateien genauer angesehen haben, lernen wir in dieser Lektion, wie Befehle als Prozesse ausgeführt werden. Darber hinaus geht es um Systemmeldungen, die Verwendung des Kernel Ring Buffers und wie `systemd` und seine Journal-Daemon (`journald`) die bis dahin übliche Systemprotokollierung verändert haben.

Prozesse

Jedes Mal, wenn ein Benutzer einen Befehl ausführt, werden ein Programm ausgeführt und ein oder mehrere Prozesse generiert.

Prozesse sind hierarchisch geordnet. Nachdem der Kernel beim Booten in den Speicher geladen wurde, wird der erste Prozess gestartet, der wiederum andere Prozesse startet, die wiederum andere Prozesse starten können. Jeder Prozess hat eine eindeutige Kennung (PID) und eine Kennung des Elternprozesses (Parent Process, PPID) — positive ganze, fortlaufende Zahlen.

Prozesse dynamisch untersuchen: `top`

Mit dem Befehl `top` erhalten Sie eine dynamische Liste aller laufenden Prozesse:

```
$ top

top - 11:10:29 up  2:21,  1 user,  load average: 0,11, 0,20, 0,14
Tasks:  73 total,   1 running,  72 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,3 sy,  0,0 ni, 99,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :  1020332 total,   909492 free,    38796 used,    72044 buff/cache
KiB Swap:  1046524 total,  1046524 free,        0 used.   873264 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
   436 carol     20   0   42696   3624   3060 R  0,7  0,4   0:00.30 top
     4 root      20   0       0      0      0 S  0,3  0,0   0:00.12 kworker/0:0
   399 root      20   0   95204   6748   5780 S  0,3  0,7   0:00.22 sshd
     1 root      20   0   56872   6596   5208 S  0,0  0,6   0:01.29 systemd
     2 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kthreadd
     3 root      20   0       0      0      0 S  0,0  0,0   0:00.02 ksoftirqd/0
     5 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 kworker/0:0H
     6 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kworker/u2:0
     7 root      20   0       0      0      0 S  0,0  0,0   0:00.08 rcu_sched
     8 root      20   0       0      0      0 S  0,0  0,0   0:00.00 rcu_bh
     9 root      rt   0       0      0      0 S  0,0  0,0   0:00.00 migration/0
    10 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 lru-add-drain
    (...)
```

Wie wir oben sehen, liefert `top` auch Informationen über den Speicher- und CPU-Verbrauch des Gesamtsystems sowie für jeden Prozess.

`top` erlaubt dem Benutzer eine gewisse Interaktion.

Standardmäßig ist die Ausgabe nach dem Prozentsatz der von jedem Prozess verbrauchten CPU-Zeit in absteigender Reihenfolge sortiert, was durch Drücken der folgenden Tasten innerhalb von `top` zu ändern ist:

`M`

Sortieren nach Speicherverbrauch.

`N`

Sortieren nach Prozess-ID.

`T`

Sortieren nach Laufzeit.

`P`

Sortieren nach Prozentsatz der CPU-Auslastung.

Um zwischen absteigender/aufsteigender Reihenfolge zu wechseln, drücken Sie einfach `R`.

TIP

Eine hübschere und benutzerfreundlichere Version von `top` ist `htop`. Eine weitere — vielleicht schon zu ausführliche — Alternative ist `atop`. Wenn nicht bereits in Ihrem System installiert, nutzen Sie den Paketmanager, um beide zu installieren und auszuprobieren.

Eine Momentaufnahme von Prozessen: `ps`

Ein weiterer sehr nützlicher Befehl für Informationen über Prozesse ist `ps`. Während `top` dynamische Informationen liefert, ist `ps` statisch.

Ohne Optionen aufgerufen, ist die Ausgabe von `ps` ziemlich übersichtlich und bezieht sich nur auf die Prozesse innerhalb der aktuellen Shell:

```
$ ps
  PID TTY          TIME CMD
 2318 pts/0    00:00:00 bash
 2443 pts/0    00:00:00 ps
```

Die angezeigten Informationen umfassen die Prozesskennung (`PID`), das Terminal, in dem der Prozess ausgeführt wird (`TTY`), die vom Prozess benötigte CPU-Zeit (`TIME`) und den Befehl zum Starten des Prozesses (`CMD`).

Ein nützlicher Schalter für `ps` ist `-f`, der eine vollständige Liste anzeigt:

```
$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
carol     2318  1682  0 08:38 pts/1    00:00:00 bash
carol     2443  2318  0 08:46 pts/1    00:00:00 ps -f
```

In Verbindung mit anderen Schaltern zeigt `-f` die Beziehung zwischen Eltern- und Kindprozessen an:

```
$ ps -uf
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
carol     2318  0.0  0.1  21336  5140 pts/1    Ss   08:38   0:00 bash
carol     2492  0.0  0.0  38304  3332 pts/1    R+   08:51   0:00  \_ ps -uf
carol     1780  0.0  0.1  21440  5412 pts/0    Ss   08:28   0:00 bash
carol     2291  0.0  0.7 305352 28736 pts/0    Sl+  08:35   0:00  \_ emacs index.en.adoc
-nw
(...)
```

Ebenso kann `ps` den Prozentsatz des Speicherverbrauchs anzeigen, wenn er mit dem Schalter `-v` aufgerufen wird:

```
$ ps -v
  PID TTY      STAT   TIME  MAJFL   TRS   DRS   RSS %MEM COMMAND
 1163 tty2     Ssl+   0:00      1    67 201224 5576  0.1 /usr/lib/gdm3/gdm-x-session (...)
 (...)
```

NOTE Ein weiterer optisch ansprechender Befehl, der die Hierarchie der Prozesse darstellt, ist `pstree`, den alle gängigen Distributionen enthalten.

Prozessinformationen im Verzeichnis `/proc`

Wir haben das Dateisystem `/proc` bereits kennengelernt. `/proc` enthält ein nummeriertes Unterverzeichnis für jeden laufenden Prozess im System (die Nummer ist die `PID` des Prozesses):

```
carol@debian:~# ls /proc
1    108  13   17   21   27   354  41   665  8    9
10   109  14   173  22   28   355  42   7    804  915
103  11   140  18   23   29   356  428  749  810  918
104  111  148  181  24   3    367  432  75   811
105  112  149  19   244  349  370  433  768  83
106  115  15   195  25   350  371  5    797  838
107  12   16   2    26   353  404  507  798  899
(...)
```

Sämtliche Informationen zu einem bestimmten Prozess liegen also in einem Verzeichnis. Lassen Sie uns den Inhalt des ersten Prozesses auflisten, dessen `PID` gleich `1` ist (die Ausgabe wurde zur besseren Lesbarkeit abgeschnitten):

```
# ls /proc/1/
```

```
attr        cmdline          environ  io        mem       ns
autogroup   comm             exe      limits    mountinfo numa_maps
auxv        coredump_filter  fd       loginuid  mounts    oom_adj
...
```

Sie können z.B. die ausführbare Datei des Prozesses überprüfen:

```
# cat /proc/1/cmdline; echo
/sbin/init
```

Wie Sie sehen, ist die Binärdatei, die die Hierarchie der Prozesse gestartet hat, `/sbin/init`.

NOTE Befehle können mit einem Semikolon (`;`) verknüpft werden. Der Grund für die Verwendung des Befehls `echo` ist die Bereitstellung einer neuen Zeile. Führen Sie einfach `cat /proc/1/cmdline` aus, um den Unterschied zu sehen.

Die Systemlast

Jeder Prozess in einem System verbraucht potenziell Systemressourcen. Die so genannte *Systemlast* (*System Load*) versucht, die Gesamtlast des Systems zu einem einzigen numerischen Indikator zu aggregieren, den Sie mit dem Befehl `uptime` sehen:

```
$ uptime
 22:12:54 up 13 days, 20:26,  1 user,  load average: 2.91, 1.59, 0.39
```

Die drei letzten Ziffern zeigen die durchschnittliche Load des Systems für die letzte Minute (`2.91`), die letzten fünf Minuten (`1.59`) und die letzten fünfzehn Minuten (`0.39`).

Jede dieser Zahlen gibt an, wie viele Prozesse entweder auf CPU-Ressourcen oder auf den Abschluss von Ein-/Ausgabevorgängen gewartet haben, d.h. diese Prozesse waren ausführbereit, wenn sie die entsprechenden Ressourcen bekommen hätten.

Systemprotokoll und Systemmeldungen

Sobald der Kernel und die Prozesse mit der Ausführung und Kommunikation untereinander beginnen, entstehen viele Informationen. Die meisten werden an Dateien gesendet - die sogenannten *Logfiles* oder einfach *Logs*.

Ohne Logging wäre die Suche nach Ereignissen auf einem Server für Systemadministratoren überaus schwierig. Darum ist es wichtig, alle Systemereignisse standardisiert und zentralisiert zu

erfassen und im Auge zu behalten. Protokolle sind entscheidend, wenn es um Fehlersuche und Sicherheit geht, aber auch um verlässliche Datenquellen zum Verständnis von Systemstatistiken und zur Vorhersage weiterer Entwicklungen.

Logging mit dem syslog-Daemon

Traditionell werden Systemmeldungen von dem Logging-Tool syslog oder einem der davon abgeleiteten Tools wie syslog-ng oder rsyslog verwaltet. Der Logging-Daemon sammelt Nachrichten von anderen Diensten und Programmen und speichert sie in Protokolldateien, typischerweise unter `/var/log`. Einige Dienste kümmern sich jedoch um ihre eigenen Protokolle (z.B. der Apache HTTPD Webserver), ebenso wie der Linux-Kernel einen In-Memory-Ringpuffer zur Speicherung seiner Log-Nachrichten verwendet.

Log-Dateien in `/var/log`

Da es sich bei Logs um Daten handelt, die sich im Laufe der Zeit verändern, finden Sie sie normalerweise in `/var/log`.

Wenn Sie sich `/var/log` ansehen, werden Sie feststellen, dass die Namen der Protokolle — bis zu einem gewissen Grad — selbsterklärend sind. Hier einige Beispiele:

`/var/log/auth.log`
: Speichert Informationen zur Authentifizierung.

`/var/log/kern.log`
: Speichert Kernel-Informationen.

`/var/log/syslog`
: Speichert Systeminformationen.

`/var/log/messages`
: Speichert System- und Anwendungsdaten.

> **NOTE** Der genaue Name und Inhalt der Protokolldateien kann je nach Linux-Distribution variieren.

Zugriff auf Protokolldateien

Denken Sie beim Durchsuchen von Protokolldateien daran, root zu sein (wenn Sie keine Leseberechtigung haben) und einen Pager wie `less` zu verwenden:

```
# less /var/log/messages
Jun  4 18:22:48 debian liblogging-stdlog:  [origin software="rsyslogd" swVersion="8.24.0" x-
pid="285" x-info="http://www.rsyslog.com"] rsyslogd was HUPed
Jun 29 16:57:10 debian kernel: [    0.000000] Linux version 4.9.0-8-amd64 (debian-
kernel@lists.debian.org) (gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9u1) ) #1 SMP
Debian 4.9.130-2 (2018-10-27)
Jun 29 16:57:10 debian kernel: [    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-
8-amd64 root=/dev/sda1 ro quiet
```

Alternativ können Sie `tail` mit dem Schalter `-f` verwenden, um die neuesten Nachrichten der Datei zu lesen und dynamisch neue Zeilen anzuzeigen, während sie angehängt werden:

```
# tail -f /var/log/messages
Jul  9 18:39:37 debian kernel: [    2.350572] RAPL PMU: hw unit of domain psys 2^-0 Joules
Jul  9 18:39:37 debian kernel: [    2.512802] input: VirtualBox USB Tablet as
/devices/pci0000:00/0000:00:06.0/usb1/1-1/1-1:1.0/0003:80EE:0021.0001/input/input7
Jul  9 18:39:37 debian kernel: [    2.513861] Adding 1046524k swap on /dev/sda5.  Priority:-
1 extents:1 across:1046524k FS
Jul  9 18:39:37 debian kernel: [    2.519301] hid-generic 0003:80EE:0021.0001:
input,hidraw0: USB HID v1.10 Mouse [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0
Jul  9 18:39:37 debian kernel: [    2.623947] snd_intel8x0 0000:00:05.0: white list rate for
1028:0177 is 48000
Jul  9 18:39:37 debian kernel: [    2.914805] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not
ready
Jul  9 18:39:39 debian kernel: [    4.937283] e1000: enp0s3 NIC Link is Up 1000 Mbps Full
Duplex, Flow Control: RX
Jul  9 18:39:39 debian kernel: [    4.938493] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link
becomes ready
Jul  9 18:39:40 debian kernel: [    5.315603] random: crng init done
Jul  9 18:39:40 debian kernel: [    5.315608] random: 7 urandom warning(s) missed due to
ratelimiting
```

Sie finden die Ausgabe im folgenden Format:

- Zeitstempel
- Hostname, von dem die Nachricht kam
- Name des Programms/Dienstes, der die Nachricht erzeugte
- PID des Programms, das die Nachricht erzeugte
- Beschreibung der Aktion, die stattgefunden hat

Die meisten Logfiles sind Klartextdateien, aber einige wenige können binäre Daten enthalten, wie z.B. `/var/log/wtmp` mit Daten, die für eine erfolgreiche Anmeldungen relevant sind. Der Befehl `file` gibt Aufschluss:

```
$ file /var/log/wtmp
/var/log/wtmp: dBase III DBT, version number 0, next free block index 8
```

Solche Dateien werden normalerweise mit speziellen Befehlen gelesen. `last` wird verwendet, um die Daten in `/var/log/wtmp` zu interpretieren:

```
$ last
carol    tty2         :0               Thu May 30 10:53   still logged in
reboot   system boot  4.9.0-9-amd64    Thu May 30 10:52   still running
carol    tty2         :0               Thu May 30 10:47 - crash  (00:05)
reboot   system boot  4.9.0-9-amd64    Thu May 30 09:11   still running
carol    tty2         :0               Tue May 28 08:28 - 14:11  (05:42)
reboot   system boot  4.9.0-9-amd64    Tue May 28 08:27 - 14:11  (05:43)
carol    tty2         :0               Mon May 27 19:40 - 19:52  (00:11)
reboot   system boot  4.9.0-9-amd64    Mon May 27 19:38 - 19:52  (00:13)
carol    tty2         :0               Mon May 27 19:35 - down   (00:03)
reboot   system boot  4.9.0-9-amd64    Mon May 27 19:34 - 19:38  (00:04)
```

NOTE Ähnlich wie bei `/var/log/wtmp` speichert `/var/log/btmp` Informationen über fehlgeschlagene Anmeldeversuche, und der spezielle Befehl zum Lesen des Inhalts ist `lastb`.

Log Rotation

Protokolldateien können über Wochen oder Monate hinweg stark wachsen und den gesamten freien Festplattenspeicher beanspruchen. Hier hilft `logrotate`, das eine Log Rotation oder einen Zyklus implementiert, so dass Log-Dateien umbenannt, archiviert und/oder komprimiert, manchmal per E-Mail an den Systemadministrator gesendet und schließlich gelöscht werden, wenn sie ein bestimmtes Alter erreicht haben. Die Konventionen zur Benennung dieser rotierten Logfiles sind vielfältig (etwa das Anfügen eines Suffixes mit Datum), aber das einfache Hinzufügen eines Suffixes mit einer ganzen Zahl ist üblich:

```
# ls /var/log/apache2/
access.log  error.log  error.log.1  error.log.2.gz  other_vhosts_access.log
```

Beachten Sie, dass `error.log.2.gz` bereits mit `gzip` komprimiert wurde (daher das Suffix `.gz`).

Der Kernel Ring Buffer

Der Kernel Ring Buffer ist eine Datenstruktur fester Größe, die Kernel-Meldungen sowohl beim Boot-Prozess als auch live aufzeichnet. Eine wichtige Funktion besteht darin, alle beim Booten erzeugten Kernel-Meldungen zu protokollieren, solange `syslog` noch nicht verfügbar ist. Der Befehl `dmesg` gibt den Kernel Ring Buffer aus (der früher auch in `/var/log/dmesg` gespeichert war). Aufgrund der Erweiterung des Ringspeichers wird dieser Befehl normalerweise in Kombination mit dem Textfilterprogramm `grep` oder einem Pager wie `less` verwendet. Um etwa nach Boot-Meldungen zu suchen:

```
$ dmesg | grep boot
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64 root=UUID=5216e1e4-ae0e-
441f-b8f5-8061c0034c74 ro quiet
[    0.000000] smpboot: Allowing 1 CPUs, 0 hotplug CPUs
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64
root=UUID=5216e1e4-ae0e-441f-b8f5-8061c0034c74 ro quiet
[    0.144986] AppArmor: AppArmor disabled by boot time parameter
(...)
```

NOTE Wenn der Kernel Ring Buffer stetig mit neuen Nachrichten wächst, verschwinden die ältesten.

Das System-Journal: systemd-journald

Seit 2015 ersetzt systemd SysV Init als *de facto* System- und Servicemanager in den meisten großen Linux-Distributionen, so dass der Journal-Daemon (journald) zur Standard-Log-Komponente geworden ist und Syslog weitestgehend ablöst. Die Daten werden nicht mehr im Klartext, sondern in Binärform gespeichert, so dass das Dienstprogramm `journalctl` zum Lesen der Protokolle erforderlich ist. Darüber hinaus ist journald syslog-kompatibel und kann in syslog integriert werden.

`journalctl` ist das Dienstprogramm zum Lesen und Abfragen der Journal-Datenbank von systemd. Ohne Optionen aufgerufen, gibt es das gesamte Journal aus:

```
# journalctl
-- Logs begin at Tue 2019-06-04 17:49:40 CEST, end at Tue 2019-06-04 18:13:10 CEST. --
jun 04 17:49:40 debian systemd-journald[339]: Runtime journal (/run/log/journal/) is 8.0M,
max 159.6M, 151.6M free.
jun 04 17:49:40 debian kernel: microcode: microcode updated early to revision 0xcc, date =
2019-04-01
Jun 04 17:49:40 debian kernel: Linux version 4.9.0-8-amd64 (debian-kernel@lists.debian.org)
```

```
(gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9u1) )
Jun 04 17:49:40 debian kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-8-amd64
root=/dev/sda1 ro quiet
(...)
```

Wenn Sie es mit den Schaltern `-k` oder `--dmesg` aufrufen, entspricht es dem Aufruf des Befehls `dmesg`:

```
# journalctl -k
[    0.000000] Linux version 4.9.0-9-amd64 (debian-kernel@lists.debian.org) (gcc version
6.3.0 20170516 (Debian 6.3.0-18+deb9u1) ) #1 SMP Debian 4.9.168-1+deb9u2 (2019-05-13)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.9.0-9-amd64 root=UUID=5216e1e4-ae0e-
441f-b8f5-8061c0034c74 ro quiet
(...)
```

Weitere interessante Optionen für `journalctl` sind:

`-b`, **`--boot`**

Zeigt Boot-Informationen an.

`-u`

Zeigt Meldungen über eine bestimmte Einheit an, wobei eine Einheit grob als jede vom System verwaltete Ressource definiert werden kann. So wird `journalctl -u apache2.service` verwendet, um Meldungen über den `apache2` Webserver zu lesen.

`-f`

Zeigt die neuesten Journal-Einträge an und gibt immer wieder neue Einträge aus, sobald sie an das Journal angehängt werden — ähnlich `tail -f`.

Geführte Übungen

1. Werfen Sie einen Blick auf die folgende Ausgabe von `top` und beantworten Sie die folgenden Fragen:

```
carol@debian:~$ top

top - 13:39:16 up 31 min,  1 user,  load average: 0.12, 0.15, 0.10
Tasks:  73 total,   2 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,  0.4 sy,  0.0 ni, 98.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1020332 total,   698700 free,   170664 used,   150968 buff/cache
KiB Swap:  1046524 total,  1046524 free,        0 used.   710956 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  605 nobody    20   0 1137620 132424  34256 S  6.3 13.0   1:47.24 ntopng
  444 www-data  20   0  364780   4132   2572 S  0.3  0.4   0:00.44 apache2
  734 root      20   0   95212   7004   6036 S  0.3  0.7   0:00.36 sshd
  887 carol     20   0   46608   3680   3104 R  0.3  0.4   0:00.03 top
    1 root      20   0   56988   6688   5240 S  0.0  0.7   0:00.42 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.09 ksoftirqd/0
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.87 kworker/0:0
(...)
```

   - Welche Prozesse wurden vom Benutzer `carol` gestartet?

   - Welches virtuelle Verzeichnis von `/proc` sollten Sie besuchen, um nach Daten des Befehls `top` zu suchen?

   - Welcher Prozess wurde zuerst ausgeführt? Woher wissen Sie das?

   - Vervollständigen Sie die Tabelle und geben Sie an, in welchem Bereich der `top`-Ausgabe die folgenden Informationen zu finden sind:

Information zu	**Übersichtsbereich**	**Aufgabenbereich**
Speicher		
Swap		

Information zu	**Übersichtsbereich**	**Aufgabenbereich**
PID		
CPU-Zeit		
Befehle		

2. Mit welchem Befehl werden die folgenden binären Protokolle gelesen?
   - `/var/log/wtmp`
   - `/var/log/btmp`
   - `/run/log/journal/2a7d9730cd3142f4b15e20d6be631836/system.journal`
3. Welche Befehle würden Sie in Kombination mit `grep` verwenden, um die folgenden Informationen über Ihr Linux-System zu erhalten?
   - Wann wurde das System zuletzt neu gestartet (`wtmp`)?
   - Welche Festplatten sind installiert (`kern.log`)?
   - Wenn erfolgte die letzte Anmeldung (`auth.log`)?
4. Welche zwei Befehle würden Sie verwenden, um den Kernel Ring Buffer anzuzeigen?
5. Geben Sie an, wo die folgenden Protokollmeldungen hingehören:
   - `Jul 10 13:37:39 debian dbus[303]: [system] Successfully activated service 'org.freedesktop.nm_dispatcher'`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 11:23:58 debian kernel: [ 1.923349] usbhid: USB HID core driver`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 14:02:53 debian sudo: pam_unix(sudo:session): session opened for user root by carol(uid=0)`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 11:23:58 debian NetworkManager[322]: <info> [1562750638.8672] NetworkManager (version 1.6.2) is starting…`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

6. Hat `journalctl` Informationen über die folgenden Einheiten?

Einheit	**Befehl**
`ssh`	
`networking`	
`rsyslog`	
`cron`	

Offene Übungen

1. Betrachten Sie die Ausgabe von `top` in der geführten Übung und beantworten Sie die folgenden Fragen:
   - Welche zwei Schritte würden folgen, um den *apache* Webserver zu töten?
   - Wie könnten Sie im Übersichtsbereich die Informationen über den physischen Speicher und den Swap mit Fortschrittsbalken anzeigen?
   - Sortieren Sie nun die Prozesse nach Speicherverbrauch:
   - Da nun Speicherinformationen in Fortschrittsbalken angezeigt werden und die Prozesse nach Speichernutzung sortiert sind, speichern Sie diese Konfiguration, so dass Sie sie beim nächsten Aufruf von `top` als Standard verwendet wird:
   - Welche Datei speichert die Konfigurationseinstellungen von `top`? Wo liegt sie? Wie kann prüfen, dass es sie gibt?
2. Machen Sie sich mit dem Befehl `exec` in Bash vertraut. Starten Sie eine Bash-Sitzung, finden Sie den Bash-Prozess mit `ps`, führen Sie anschließend `exec /bin/sh` aus und suchen Sie dann erneut nach dem Prozess mit derselben PID.
3. Folgen Sie diesen Schritten, um Kernel-Ereignisse und die dynamische Verwaltung von Geräten durch udev zu untersuchen:
   - Schließen Sie ein USB-Laufwerk direkt an Ihren Computer an (Hotplug). Führen Sie `dmesg` aus und achten Sie auf die letzten Zeilen. Wie lautet ist die jüngste Zeile?
   - Führen Sie unter Berücksichtigung der Ausgabe des vorherigen Befehls `ls /dev/sd*` aus und stellen Sie sicher, dass Ihr USB-Stick in der Liste erscheint. Wie lautet die Ausgabe?
   - Entfernen Sie nun das USB-Laufwerk und führen Sie `dmesg` erneut aus. Wie lautet die letzte Zeile?

- Führen Sie `ls /dev/sd*` erneut aus und stellen Sie sicher, dass Ihr Gerät aus der Liste verschwunden ist. Was lautet die Ausgabe?

Zusammenfassung

Im Zusammenhang mit der Datenspeicherung wurden in dieser Lektion folgende Themen behandelt: Prozessmanagement sowie Systemprotokollierung und -benachrichtigungen.

Was das Prozessmanagement betrifft, so haben wir folgendes gelernt:

- Programme erzeugen Prozesse und Prozesse existieren in einer Hierarchie.
- Jeder Prozess hat eine eindeutige Kennung (PID) und eine übergeordnete Prozesskennung (`PPID`).
- `top` ist ein sehr nützlicher Befehl, um dynamisch und interaktiv die laufenden Prozesse des Systems zu untersuchen.
- `ps` liefert eine Momentaufnahme der aktuellen laufenden Prozesse im System.
- Das Verzeichnis `/proc` enthält Verzeichnisse für jeden laufenden Prozess im System, benannt nach der jeweiligen PID.
- Das Konzept der durchschnittlichen Systemlast — was sehr nützlich ist, um die CPU-Auslastung zu überprüfen.

In Bezug auf das System-Logging sollten Sie sich merken:

- Ein Log ist eine Datei, in der Systemereignisse aufgezeichnet werden. Logs sind für die Fehlerbehebung unverzichtbar.
- Das Logging wurde traditionell von speziellen Diensten wie syslog, syslog-ng oder rsyslog durchgeführt. Dennoch verwenden einige Programme ihre eigenen Logging-Daemonen.
- Da Logs variable Daten sind, werden sie in `/var` abgelegt. Manchmal geben ihre Namen einen Hinweis auf den Inhalt (`kern.log`, `auth.log`, etc.)
- Die meisten Logs sind Klartextdateien und können mit jedem Texteditor gelesen werden, solange Sie die notwendigen Berechtigungen haben. Einige davon sind jedoch binär und müssen mit speziellen Befehlen gelesen werden.
- Um Probleme mit dem Festplattenspeicher zu vermeiden, wird die Log Rotation vom Programm `logrotate` durchgeführt.
- Der Kernel nutzt eine ringförmige Datenstruktur, den Ring Buffer, in dem Boot-Meldungen gespeichert werden (alte Nachrichten verschwinden mit der Zeit).
- Der System- und Servicemanagersystemd hat System V init in praktisch allen Distributionen abgelöst, wobei journald zum Standard-Logging-Service wurde.
- Um das Journal von systemd zu lesen, wird das Programm `journalalctl` benötigt.

Befehle, die in dieser Lektion verwendet wurden:

`cat`

Dateiinhalt verketten/ausgeben.

`dmesg`

Gibt den Kernel Ring Buffer aus.

`echo`

Zeigt eine Textzeile oder eine neue Zeile an.

`file`

Bestimmt den Dateityp.

`grep`

Gibt Zeilen aus, die einem Muster entsprechen.

`last`

Gibt eine Liste der zuletzt angemeldeten Benutzer aus.

`less`

Zeigt den Inhalt einer Datei seitenweise an.

`ls`

Listet Verzeichnisinhalte auf.

`journalctl`

Fragt das `systemd`-Journal ab.

`tail`

Zeigt die letzten Zeilen einer Datei an.

Lösungen zu den geführten Übungen

1. Werfen Sie einen Blick auf die folgende Ausgabe von `top` und beantworten Sie die folgenden Fragen:

```
carol@debian:~$ top

top - 13:39:16 up 31 min,  1 user,  load average: 0.12, 0.15, 0.10
Tasks:  73 total,   2 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,  0.4 sy,  0.0 ni, 98.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1020332 total,   698700 free,   170664 used,   150968 buff/cache
KiB Swap:  1046524 total,  1046524 free,        0 used.   710956 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  605 nobody    20   0 1137620 132424  34256 S  6.3 13.0   1:47.24 ntopng
  444 www-data  20   0  364780   4132   2572 S  0.3  0.4   0:00.44 apache2
  734 root      20   0   95212   7004   6036 S  0.3  0.7   0:00.36 sshd
  887 carol     20   0   46608   3680   3104 R  0.3  0.4   0:00.03 top
    1 root      20   0   56988   6688   5240 S  0.0  0.7   0:00.42 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.09 ksoftirqd/0
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.87 kworker/0:0
(...)
```

   - Welche Prozesse wurden vom Benutzer `carol` gestartet?

     Lösung: Nur einer: `top`.

   - Welches virtuelle Verzeichnis von `/proc` sollten Sie besuchen, um nach Daten des Befehls `top` zu suchen?

     Lösung: `/proc/887`

   - Welcher Prozess wurde zuerst ausgeführt? Woher wissen Sie das?

     Lösung: `systemd`, weil es die `PID` `1` hat.

   - Vervollständigen Sie die Tabelle und geben Sie an, in welchem Bereich der `top`-Ausgabe die folgenden Informationen zu finden sind:

Information zu	**Übersichtsbereich**	**Aufgabenbereich**
Speicher	Ja	Ja

Information zu	Übersichtsbereich	Aufgabenbereich
Swap	Ja	Nein
PID	Nein	Ja
CPU-Zeit	Ja	Ja
Befehle	Nein	Ja

2. Mit welchem Befehl werden die folgenden binären Protokolle gelesen?
   - `/var/log/wtmp`

     Lösung: `last`
   - `/var/log/btmp`

     Lösung: `lastb`
   - `/run/log/journal/2a7d9730cd3142f4b15e20d6be631836/system.journal`

     Lösung: `journalctl`
3. Welche Befehle würden Sie in Kombination mit `grep` verwenden, um die folgenden Informationen über Ihr Linux-System zu erhalten?
   - Wann wurde das System zuletzt neu gestartet (`wtmp`)?

     Lösung: `last`
   - Welche Festplatten sind installiert (`kern.log`)?

     Lösung: `less /var/log/kern.log`
   - Wenn erfolgte die letzte Anmeldung (`auth.log`)?

     Lösung: `less /var/log/auth.log`
4. Welche zwei Befehle würden Sie verwenden, um den Kernel Ring Buffer anzuzeigen?

   `dmesg` und `journalctl -k` (oder auch `journalctl --dmesg`).
5. Geben Sie an, wo die folgenden Protokollmeldungen hingehören:
   - `Jul 10 13:37:39 debian dbus[303]: [system] Successfully activated service 'org.freedesktop.nm_dispatcher'`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	X
`/var/log/messages`	

- `Jul 10 11:23:58 debian kernel: [ 1.923349] usbhid: USB HID core driver`

`/var/log/auth.log`	
`/var/log/kern.log`	X
`/var/log/syslog`	
`/var/log/messages`	X

`Jul 10 14:02:53 debian sudo: pam_unix(sudo:session): session opened for user root by carol(uid=0)`

`/var/log/auth.log`	X
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	

- `Jul 10 11:23:58 debian NetworkManager[322]: <info> [1562750638.8672] NetworkManager (version 1.6.2) is starting…`

`/var/log/auth.log`	
`/var/log/kern.log`	
`/var/log/syslog`	
`/var/log/messages`	X

6. Hat `journalctl` Informationen über die folgenden Einheiten?

Einheit	**Befehl**
`ssh`	`journalctl -u ssh.service`
`networking`	`journalctl -u networking.service`
`rsyslog`	`journalctl -u rsyslog.service`

Einheit	Befehl
`cron`	`journalctl -u cron.service`

Lösungen zu den offenen Übungen

1. Betrachten Sie die Ausgabe von `top` in der geführten Übung und beantworten Sie die folgenden Fragen:
   - Welche zwei Schritte würden folgen, um den *apache* Webserver zu töten?

     Zuerst drücken Sie `k`, dann geben Sie einen `kill`-Wert ein.

   - Wie könnten Sie im Übersichtsbereich die Informationen über den physischen Speicher und den Swap mit Fortschrittsbalken anzeigen?

     Durch ein- oder zweimaliges Drücken von `m.

   - Sortieren Sie nun die Prozesse nach Speicherverbrauch:

     `M`

   - Da nun Speicherinformationen in Fortschrittsbalken angezeigt werden und die Prozesse nach Speichernutzung sortiert sind, speichern Sie diese Konfiguration, so dass Sie sie beim nächsten Aufruf von `top` als Standard verwendet wird:

     `W`

   - Welche Datei speichert die Konfigurationseinstellungen von `top`? Wo liegt sie? Wie kann prüfen, dass es sie gibt?

     Die Datei ist `~/.config/procps/toprc` und befindet sich im Heimatverzeichnis des Benutzers (`~`). Da es sich um eine versteckte Datei handelt (sie beginnt mit einem Punkt), können wir mit `ls -a` (Liste aller Dateien) auf ihre Existenz prüfen. Diese Datei kann durch Drücken von `Shift` + `W` innerhalb `top` erzeugt werden.

2. Machen Sie sich mit dem Befehl `exec` in Bash vertraut. Starten Sie eine Bash-Sitzung, finden Sie den Bash-Prozess mit `ps`, führen Sie anschließend `exec /bin/sh` aus und suchen Sie dann erneut nach dem Prozess mit derselben PID.

   `exec` ersetzt einen Prozess durch einen anderen Befehl. Im folgenden Beispiel sehen wir, dass der Bash-Prozess durch `/bin/sh` ersetzt wird (anstatt `/bin/sh` zu einem Unterprozess zu werden):

```
$ echo $$
19877
$ ps auxf | grep 19877 | head -1
```

```
carol   19877  0.0  0.0   7448  3984 pts/25   Ss   21:17   0:00  \_ bash
$ exec /bin/sh
sh-5.0$ ps auxf | grep 19877 | head -1
carol   19877  0.0  0.0   7448  3896 pts/25   Ss   21:17   0:00  \_ /bin/sh
```

3. Folgen Sie diesen Schritten, um Kernel-Ereignisse und die dynamische Verwaltung von Geräten durch udev zu untersuchen:
   - Schließen Sie ein USB-Laufwerk direkt an Ihren Computer an (Hotplug). Führen Sie `dmesg` aus und achten Sie auf die letzten Zeilen. Wie lautet ist die jüngste Zeile?

     Sie sollten etwas wie `[ 1967.700468] sd 6:0:0:0: [sdb] Attached SCSI removable disk` erhalten.

   - Führen Sie unter Berücksichtigung der Ausgabe des vorherigen Befehls `ls /dev/sd*` aus und stellen Sie sicher, dass Ihr USB-Stick in der Liste erscheint. Wie lautet die Ausgabe?

     Abhängig von der Anzahl der an Ihr System angeschlossenen Geräte sollten Sie so etwas wie `/dev/sda /dev/sda /dev/sda1 /dev/sdb /dev/sdb1 /dev/sdb2` erhalten. In unserem Fall finden wir unseren USB-Stick (`/dev/sdb`) und seine beiden Partitionen (`/dev/sdb1` und `/dev/sdb2`).

   - Entfernen Sie nun das USB-Laufwerk und führen Sie `dmesg` erneut aus. Wie lautet die letzte Zeile?

     Sie sollten etwas wie `[ 2458.881695] usb 1-9: USB disconnect, device number 6` erhalten.

   - Führen Sie `ls /dev/sd*` erneut aus und stellen Sie sicher, dass Ihr Gerät aus der Liste verschwunden ist. Was lautet die Ausgabe?

     In unserem Fall: `/dev/sda /dev/sda1`.