Grundkurs Linux

Release 0.2.7

Aktualisiert am 08.08.2017

Bernhard Grotz

http://www.grund-wissen.de

Die Quelldateien dieses Buchs wurden unter Linux mittels Vim und Sphinx, die enthaltenen Graphiken mittels Inkscape erstellt. Der Quellcode sowie die Original-Graphiken können über die Projektseite heruntergeladen werden:

http://www.grund-wissen.de

Bei Fragen, Anmerkungen und Verbesserungsvorschlägen bittet der Autor um eine kurze Email an folgende Adresse:

info@grund-wissen.de

Augsburg, den 8. August 2017.

Bernhard Grotz

Inhaltsverzeichnis

Einleitung

Linux ist ein freies Betriebsystem und entspricht dem Open-Source-Grundgedanken:

- Das System und alle verfügbaren Programme sind zum einen ohne Einschränkung kostenlos nutzbar.
- Auch der Quellcode des Systems beziehungsweise der Programme kann frei heruntergeladen und nach Belieben abgeändert werden.

Dank der Sicherheit, Stabilität und Transparenz ist Linux das wohl beliebteste und am meisten verbreitete Betriebsystem für Großrechner, Router und Webserver. Mit vorkonfigurierten Varianten wie Ubuntu oder Linux Mint sind auch „normale“ Aufgaben wie das Benutzen von Office-Programmen, das Verwalten von Emails, das Abspielen von Musik- und Videodateien, usw. auch unter Linux komfortabel und ohne Programmierkenntnisse leicht umsetzbar.

Linux bietet allerdings – im Gegensatz zu kommerziellen Systemen – stets die Möglichkeit, die Funktionsweise des Betriebsystems und seiner Programme auch genauer zu verstehen. Auch können mittels einfacher Shell-Skripte Aufgaben häufig auch automatisiert gelöst werden.

Grundgedanken von Open Source

Open Source kann als ein gemeinschaftliches Prinzip der Zusammenarbeit verstanden werden – es ist daher auch nicht an ein spezielles Betriebsystem gebunden. Zahlreiche Open-Source Programme wie beispielsweise Firefox, Thunderbird, Open Office, VLC, Gimp , und Inkscape, sind inzwischen auf vielen Betriebsystemen anzufinden.[1] Am bekanntesten wurde die Open-Source-Mentalität allerdings durch Linux, welches vollständig auf freier Software aufbaut.

Jeder darf sich freie Software und Betriebsysteme kostenlos herunterladen, benutzen und (je nach *Lizenz*) auch modifizieren. Eine Übersicht über zahlreiche Open-Source-Programme bietet – unabhängig vom Betriebsystem – beispielsweise Sourceforge. Eine Übersicht über Linux-Software bietet beispielsweise das Ubuntuusers-Wiki.

[1] Geschichtlich interessant zum Verständnis von Open Source und Linux ist auch der Essay „Die Kathedrale und der Basar“ von Eric Raymond, der letztlich zu einer quellfreien Veröffentlichung des „Netscape Navigator“ – dem Vorläufer von Mozilla Firefox – und zu einem Umdenken in der Industrie geführt hat.

Die Geschichtliche Entwicklung von Open Source und Linux

Die Open-Source-Bewegung entstand, als einige Software-Firmen damit begannen, ihren Kunden Software nicht mehr in Form von Quellcode, sondern als bereits in Maschinencode umgewandelte Programme zu verkaufen. Diese waren zwar für Computer, jedoch nicht mehr für Menschen lesbar.

Etliche Programmierer befürchteten dabei vor allem einen Kontrollverlust auf Seiten der Kunden und kritisierten den mangelnden Informationsfluss seitens der Software-Hersteller. Darüber hinaus wurde – in Kombination mit der Patentierung von „geistigem Eigentum“ – die Gleichberechtigung der Programmierer als gefährdet angesehen. Das Know-How, so fürchtete man, würde sich schnell nur auf wenige Firmen konzentrieren.

Um nun derartigen Patentierungen von Informationen und den daraus resultierenden technischen und rechtlichen Problemen entgegenzuwirken, wurden unter anderem die „Open-Source“-Bewegung und die „Free Software Foundation“ ins Leben gerufen und mit der Entwicklung gemeinschaftlicher, frei verfügbarer Software begonnen.[2]

In den 1970er und 1980er Jahren entstand so das zahlreiche freie Software umfassende GNU-Projekt, das sogar eine eigene Lizenz-Regelung (GNU General Public License (GPL)) mit sich brachte. Die Grundgedanken, die Quellcode-Bibliotheken sowie die rechtlichen Grundlagen stellten letztlich in den 1990er Jahren die Basis für das neue Betriebsystem „Linux“ dar.[3]

[2] Diese beide Initiativen werden bisweilen mit dem Begriff „FOSS“ (Free and Open Source Software) zusammengefasst.

[3] Die wohl wichtigsten Prinzipien sind in einem Aufsatz von Chistian Imhorst zusammengefasst:

> Der Programmierer freier Software verschenkt mit der GPL die Kontrolle über sein Werk, nicht aber das Werk als solches. Er behält die Autorenschaft über sein Programm. Dem Benutzer der Software wiederum werden bestimmte Freiheiten gewährt, wie die Freiheit das Werk zu modifizieren und verändert zu veröffentlichen. An diese Freiheit ist nur eine Bedingung geknüpft: Das veränderte Werk muss wieder unter der GPL stehen. (...) Freie Software soll nicht Eigentum eines Einzelnen, sondern das Eigentum von allen sein.
>
> (...)
>
> Niemand ist vom Eigentum an GPL-Software ausgeschlossen. Ihre Verbreitung kann deshalb von niemandem kontrolliert werden. Wer sie haben möchte, kann sie einfach kopieren und weitergeben, wodurch die Verfügbarkeit von GPL-Software sehr schnell wächst. Die GPL verhindert zwar, dass Menschen von dem Gebrauch freier Software ausgeschlossen werden, aber sie schließt auf der anderen Seite ebenfalls aus, dass jemand aus freier Software proprietäre macht. Niemand kann daran gehindert werden, das freie Betriebssystem GNU/Linux zu benutzen, und niemandem kann es weggenommen werden. Jeder, der GNU/Linux aus

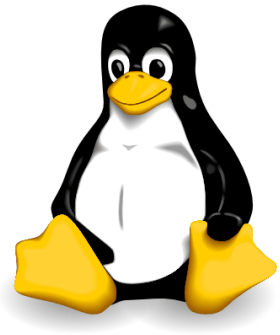


Abb. 1: Das Linux-Maskottchen Tux.

„Free Speech“ und „Free Beer“

Bei Open-Source-Software darf stets der Quellcode frei eingesehen werden; man kann derartige Software somit stets gratis herunterladen, installieren, und als Privatanwender ohne weitere Einschränkungen komplett legal nutzen.

„Freiheit“ ist allerdings ein schwer festzulegender Begriff; es gibt beispielsweise große Unterschiede zwischen einer Rede-Freiheit (Free Speech) und Freibier (Free Beer). Überträgt man diesen Vergleich auf Open-Source-Software, so kann man sagen, dass für sämtliche Open-Source-Software stets das erstere Prinzip gilt: Der Quellcode ist stets frei zugänglich, man kann also jederzeit „sehen“, wie die jeweiligen Programme funktionieren, und sie folglich auch legal herunterladen und installieren.

Auf die einzelnen Lizenzen achten muss man hingegen, wenn der Quellcode von einzelnen Programmen modifiziert beziehungsweise um zusätzliche Funktionalitäten erweitert werden soll. Der Quellcode „gehört“ einem nämlich nicht (im Sinne von „Free Beer“), so dass man diesen beispielsweise nicht einfach Code hinzufügen und das Resultat dann als eigenes Programm (womöglich sogar ohne Veröffentlichung des modifizierten Quellcodes) auf einem Webserver einsetzen oder gar verkaufen darf.

Die wichtigsten Open-Source-Lizenzen sind folgende:

- Unter einer GPL-Lizenz stehende Programme dürfen jederzeit (auch kommerziell) frei genutzt werden; Änderungen am Quellcode müssen allerdings ebenfalls wieder unter einer GPL-Lizenz stehen (Copyleft-Prinzip). GPL-Code darf zudem nicht in andere, propietäre Software eingebaut werden.

---

dem Internet herunter lädt, auf seinen Rechner installiert, Kopien davon verschenkt oder verkauft, dem gehört es auch.

In diesem Sinne ist die GPL eher eine Anti-Lizenz, weshalb Richard Stallman von ihr auch lieber als Copyleft spricht anstatt von einem Copyright.

—Christian Imhorst

Quellcode sollte nach dieser Ethik geteilt, also ohne Hindernisse ausgetauscht und für jeden frei zugänglich gemacht werden. So wurden ähnliche Lizenzen wie die Creative Commons Licencse auch für Bücher und Musik geschaffen – das berühmteste Projekt mit einer solchen Lizenz ist wohl Wikipedia.

- Unter einer LGPL beziehungsweise Apache-Lizenz stehende Programme dürfen ebenfalls frei genutzt werden; Änderungen und Erweiterungen dürfen allerdings auch unter anderen Lizenzen stehen. Diese Lizenzen weichen somit das Copyleft-Prinzip auf und erlauben auch propietäre Erweiterungen (bei denen die Nutzer dann wiederum auf die jeweiligen Lizenz-Texte achten müssen).
- Bei Werken, die unter einer Creative-Commons-Lizenz (CCL) stehen (oftmals Bücher, Texte, Bilder, usw.), gibt es verschiedene „Freiheitsgrade“, die der Urheber des Werks selbst festlegen kann. Wird beispielsweise der Zusatz „nd“ (No Derivatives“) verwendet, so darf das Werk zwar genutzt, aber nicht modifiziert werden; wird andererseitst der Zusatz „nc“ (Non-Commercial) verwendet, so wird eine kommerzielle Nutzung untersagt.

Nutzen, Verstehen, Mitmachen!

Die Linux- und Open-Source-Gemeinschaft lädt nicht nur dazu ein, eine Vielzahl an Programmen frei herunterzuladen und zu nutzen – sie ist gleichzeitig darum bemüht, den Nutzern die Vorteile von Open-Source-Projekten sowie die Funktionsweise von Software verständlich zu machen. In einem Team, in dem ein jeder von der Arbeit aller anderen profitiert, sind weitere „Kollegen“ schließlich jederzeit willkommen.. :-)

Mit Linux als Betriebsystem hat jeder Nutzer die Möglichkeit, sich den Quellcode von anderen Programmierern anzuschauen und – durch Lesen und eigenes Code-Schreiben – auch selbst ein Software-Entwickler zu werden.

Doch nicht nur Quellcode ist von Bedeutung: Alle Informationen, die unter einer gemeinnützigen Lizenz wie der Creative-Commons-License (CCL) oder der General Public License (GPL) veröffentlicht werden, tragen zum frei abrufbaren Wissensschatz und somit zum Gemeinwohl der Gesellschaft bei!

Linux installieren

Linux ist komfortabel geworden. Das gilt nicht nur für die Vielfalt und Anwenderfreundlichkeit der Programme, sondern auch für die Installations-Routinen. Dank gut ausgearbeiteter Dokumentationen (z.B. das Ubuntu-Wiki) lässt sich heute in kurzer Zeit und ohne Grundkenntnisse ein gut funktionierendes Linux-System einrichten.

Auswahl einer Linux-Distribution

Als Distribution wird im Software-Bereich eine Zusammenstellung von Software zu einem gut nutzbaren Gesamtpaket bezeichnet. Die Vielzahl an Linux-Distributionen[1] unterscheidet sich im Wesentlichen dadurch, welche Programme grundsätzlich installiert sind, wie häufig Updates erfolgen und wie viel Konfigurationen der Benutzer gewöhnlich manuell vornehmen möchte.

In Deutschland laufen die meisten Server mit der Distribution Debian; bei den Privat-PCs zählen die auf Debian basierenden Distributionen Ubuntu und Linux Mint zu den beliebtesten Systemen. Alle Tips dieser Notizen-Sammlung beziehen sich auf diese einander weitgehend ähnlichen Systeme.

[1] Je nach Vorliebe und Anwendungszweck bietet sich ein weites Spektrum an Distributionen:

- „Live"-Systeme (beispielsweise Knoppix):

  Einige Linux-Varianten sind darauf ausgelegt von einem externen Datenträger (CD, USB-Stick) geladen und ohne Installation von diesem aus genutzt zu werden.

- Extrem konfigurierbare Distributionen (beispielsweise Gentoo, Arch):

  In manchen Distributionen ist es normal, den Linux-Kernel und die Programme stets selbst aus dem Quellcode zu compilieren, d.h. in ausführbaren Maschinencode zu übersetzen. Dies bietet eine maximale Kontrolle über die laufenden Programme und kann erhebliche Geschwindigkeitsvorteile im laufenden System mit sich bringen. Um das Potential derartiger Distributionen auch ausschöpfen zu können, sollte man allerdings ausreichend mit Linux- und Hardware-Grundlagen vertraut sein.

- Distributionen von kommerziellen Anbietern (beispielsweise Red Hat Enterprise, Fedora, Mandriva):

  Diese Distributionen werden von Firmen entwickelt und gepflegt und sind insbesondere für Geschäftskunden interessant, die darauf angewiesen sind, jederzeit einen kommerziellen technischen Support in Anspruch nehmen zu können.

Wie Linux letztendlich „aussieht“, hängt nur bedingt von der Distribution ab. Die graphische Bedienoberfläche und das eigentliche Betriebsystem sind – anders als bei Windows – zweierlei Dinge. Das Betriebsystem Linux unterstützt eine Vielzahl an Bedienoberflächen („Desktop-Umgebungen“), die sich auch nach der Installation jederzeit austauschen lassen. Die wohl bekanntesten Bedienoberflächen sind GNOME und KDE. Persönlich nutze ich derzeit bevorzugt die unter Ubuntu und Linux Mint übliche Oberfläche Mate.

Die Basis-Installation

Ein Linux-System zu installieren ist heute dank moderner Hardware-Erkennung und graphischer Installations-Assistenten denkbar einfach. Das Installations-Schema ist bei fast allen Distributionen gleich:

1. Man lädt sich von der jeweiligen Homepage eine ISO-Image-Datei herunter. Persönlich bevorzuge ich derzeit Linux Mint Ubuntu Edition mit Mate-Desktop. Je nach Hardware muss die 32- oder 64-Bit-Variante gewählt werden.[2]

2. Man erstellt aus diesem Image mit einem beliebigen Brennprogramm eine bootbare CD beziehungsweise mit UNetBootin einen bootbaren USB-Stick. Eine gute Anleitung zur Erstellung eines Ubuntu-USB-Sticks, die mit Linux Mint genauso funktioniert, findet sich hier.

3. Man bootet den Rechner mit eingelegter Boot-CD beziehungsweise angestecktem Boot-USB-Stick neu.

Die Boot-Reihenfolge wird vom BIOS des Rechners festgelegt. Entsprechende Einstellungen werden im BIOS-Menü vorgenommen, das sich bei einem Neustart des Rechners meist mit `F2` (manchmal auch mit `F8`) öffnen lässt. Je nach ausgewählter Distribution erscheint automatisch ein Installations-Assistent (Debian), oder es wird ein Live-System gebootet

[2] Ältere Rechner mit einem einzelnen Prozessor (beispielsweise Intel Celeron, Intel Core2Duo, Intel Atom) benötigen ein 32-Bit-System, neuere Multi-Core-Prozessoren hingegen ein 64-Bit-System.

Ist man sich nicht sicher, welcher Systemtyp der passende ist (beispielsweise weil man nicht weiß, was für ein Prozessor eingebaut ist), so kann eine entsprechende Suchmaschinen-Anfrage weiterhelfen. Darüber hinaus kann auch ein (versehentlicher) Versuch, einen Rechner mit einem nicht passenden System zu booten, keinerlei Schaden anrichten, denn er wird unmittelbar unterbrochen und eine entsprechende Fehlermeldung ausgegeben.

(Ubuntu und Linux Mint), in dem der Installations-Assistent als Icon auf dem Desktop zu finden ist.

Zu Beginn der Installationsroutine legt man den Benutzernamen mit dazugehörigem Passwort und bevorzugte Einstellungen (Tastaturlayout, Zeitzone, etc.) fest. Nachträgliche Änderungen dieser Einstellungen sind auch später ohne Aufwand möglich.

Der wichtigste Schritt der Installation besteht anschließend in der Festlegung der zu nutzenden Festplattenpartitionen. Diese lassen sich wahlweise automatisch oder von Hand mittels eines integrierten Partitionsprogramms einrichten. Der Installations-Assistent führt die Grundinstallation daraufhin vollautomatisch durch.

Empfehlenswert: Partitionen manuell einrichten

Bei einer nicht-automatischen Festlegung der Partitionen ist, sofern genügend Festplattenspeicher vorhanden ist, folgende Partitionierung sinnvoll:

- Eine mittelgroße Partition (min. 10 GB, max. 30 GB) mit Einhängepunkt `/` für das Basis-System
- Eine beliebig große Partition (min. 10 GB) mit Einhängepunkt `/home` für eigene Dokumente, Konfigurationsdateien, etc.
- Empfehlenswert: Eine kleine Partition (2 bis 5 GB) als `swap` (Erweiterung des Arbeitsspeichers)
- Optional: Eine beliebig große Daten-Partition (ohne bei der Installation festgelegten Einhängepunkt) für gemeinsam genutzte und/oder verschlüsselte Dateien
- Optional: Nicht verwendeter Speicherplatz für ein weiteres, parallel installierbares oder bereits installiertes Betriebsystem[3]

Eine separate `/home`-Partition bietet den Vorteil, dass das System jederzeit neu installiert werden kann, ohne dass eigene Daten und Einstellungen davon berührt werden. Das gilt ebenso für Systeme mit mehreren Benutzern.

Für die `/` beziehungsweise `/home`-Partition empfiehlt sich als Dateisystem `EXT-4`, da es sehr schnell, sehr stabil und quasi wartungsfrei ist. Für Daten-Partitionen empfiehlt sich ebenfalls `EXT-4` als Dateisystem, solange man nur mit Linux darauf zugreifen möchte. Falls man die Daten auch unter Windows oder MacOS nutzen mag, muss `FAT32` als

[3] Linux lässt sich auch parallel zu einem bestehenden Windows-System installieren. Hierzu nutzt man am besten eine eigene Festplatte oder legt mit dem Installations-Assistenten eine neue `ext4`-Partition an (mindestens 15 GB) und installiert Linux in diesen Bereich; auch zwei neue Partitionen mit den Einhängepunkten `/` für das Grundsystem und `/home` für persönliche Dateien sind als Variante möglich. Nach einer üblichen Installation lässt sich anschließend bei jedem Rechnerstart in einem Menü auswählen, welches Betriebsystem gestartet werden soll.

Achtung: Bei einer Veränderung einer bestehenden Partition – beispielsweise einer Verkleinerung, um Platz für eine neute Partition zu schaffen – lässt sich ein Datenverlust niemals völlig ausschließen. Eine Sicherheitskopie bestehender Daten ist daher auf alle Fälle empfehlenswert!

Linux kann lesend und schreibend auf alle Windows-Dateien und zugreifen. Windows kann jedoch nicht mit Linux-Dateisystemen umgehen, da es beispielsweise nicht zwischen Groß- und Kleinschreibung in Dateinamen unterscheidet. Möchte man auf bestimmte Daten mit beiden Systemen zugreifen, so müssen diese folglich auf einer Windows-Partition liegen.

Dateisystem verwendet werden. `FAT32` verfügt allerdings über keine Unterstützung von nützlichen Symlinks und bietet keine Unterscheidung von Groß- und Kleinschreibung.

Passwortgeschützte Partitionen

Linux ist als Betriebsystem verhältnismäßig sicher. Lässt man jedoch beispielsweise ein Notebook unbeaufsichtigt liegen, so helfen die besten Konfigurationen nichts, um vertrauliche Daten vor unbefugtem Fremdzugriff zu schützen. Eine Festplatte kann einfach ausgebaut und extern an einen anderen PC angeschlossen werden, und schon sind alle Daten (Passwörter, evtl. Onlinebanking-Daten, Emails, etc.) frei abrufbar...

Wer private Daten in einem passwortgeschützten („verschlüsselten“) Bereich ablegen möchte, kann sich unter Linux für eine der folgende Möglichkeiten entscheiden:

***Partition-Verschlüsselung*:** Die Partition kann bereits während der Installation – ohne Festlegung eines Einhängepunktes – auf einem freien (unformatierten) Bereich eines Datenträgers angelegt werden.

Der Vorteil dieser Methode liegt darin, dass sie verhältnismäßig einfach einzurichten und die verschlüsselte Partition unabhängig vom System ist. Somit kann der geschützte Bereich auch auf einem laufenden Rechner verschlossen bleiben.

Nachteilig bei dieser Methode ist, dass jeder Unbefugte mit Hardware-Zugriff das Betriebsystem ohne Hindernis verändern kann, beispielsweise um Trojaner oder Keyboard- beziehungsweise Datenlogger zu installieren.

***System-Verschlüsselung*:** Beim Start erscheint eine Passwort-Abfrage. Erst wenn das Passwort richtig eingegeben wurde, wird die Systempartition entschlüsselt, und der Rechner bootet.

Der Vorteil dieser Methode liegt darin, dass kein Unbefugter Zugriff auf Teile des Systems oder der persönlichen Dateien hat – sofern er den Rechner ausgeschaltet vorfindet.

Nachteilig bei dieser Methode ist, dass sie einem Rechner im laufenden Betrieb – die Systempartition ist wohl immer geöffnet – keinerlei Schutz bietet. Darüber hinaus setzt diese Methode setzt einige Linux-Kenntnisse voraus und ist für Anfänger ungeeignet.

Darüber hinaus ist es möglich, mittels Truecrypt passwortgeschützte Daten-„Container“ zu erstellen. Diese können nach der Installation auf einer beliebigen Partition eingerichtet werden und sind auch auf anderen Betriebsystemen nutzbar.

Der Vorteil bei der Verwendung von `truecrypt` liegt darin, dass – im Gegensatz zu den obigen Methoden – auch Windows- und MacOS-Systeme auf den passwortgeschützten Bereich zugreifen können.

Als Nachteil ist zu nennen, dass das Erstellen eines Containers – je nach Größe und Rechnerleistung – mehrere Stunden dauern kann; `truecrypt` wird zudem nicht mehr aktiv weiterentwickelt.

Zusatzpakete installieren

Nach einer erfolgreichen Basis-Installation werden üblicherweise noch weitere Anpassungen vorgenommen. Im Wesentlichen gibt es dafür zwei Gründe:

- Einerseits mag man als Nutzer zusätzliche Software gemäß den eigenen Vorlieben installieren. Die Menge an verfügbarer Software passt allerdings nicht unbedingt auf eine CD, eine DVD, oder einen USB-Stick. Die Programme werden darüber hinaus beständig weiter entwickelt und sind möglicherweise in der Zwischenzeit aktualisiert worden.
- Andererseits werden je nach Linux-Distribution nur Software-Pakete mitgeliefert, die bestimmten Kriterien genügen (beispielsweise gewisse Lizenzbedingungen aufweisen). Manche wichtigen Pakete müssen daher, selbst wenn sie frei verfügbar sind, manuell installiert werden.

Paketverwaltung mittels graphischer Bedienoberfläche

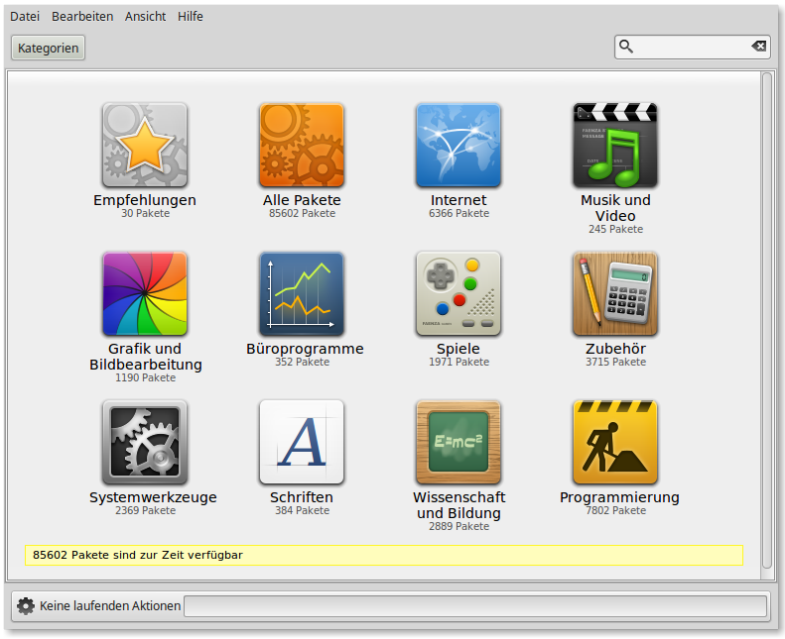
Je nach Linux-Distribution gibt es unterschiedliche graphische Verwaltungsprogramme, mit denen Programme installiert, aktualisiert oder auch wieder entfernt werden können:

- Unter Ubuntu gibt es Synaptic,
- Unter Linux Mint gibt es zusätzlich das Programm `mintinstall`, das auf Synaptic aufbaut und eine sehr komfortable (aber etwas langsame) graphische Oberfläche bietet.

Die obigen Programme sind nur nutzbar, wenn der angemeldete Benutzer sich zeitweise auch SuperUser-Rechte geben darf; gibt es nur einen Benutzer-Account, den man bei der Installation selbst eingerichtet hat, so ist dies bei diesem Account stets der Fall.[4] Man muss lediglich beim Start der Anwendungsverwaltung das eigene Benutzerpasswort eingeben, um die Anwendung im SuperUser-Modus laufen zu lassen.

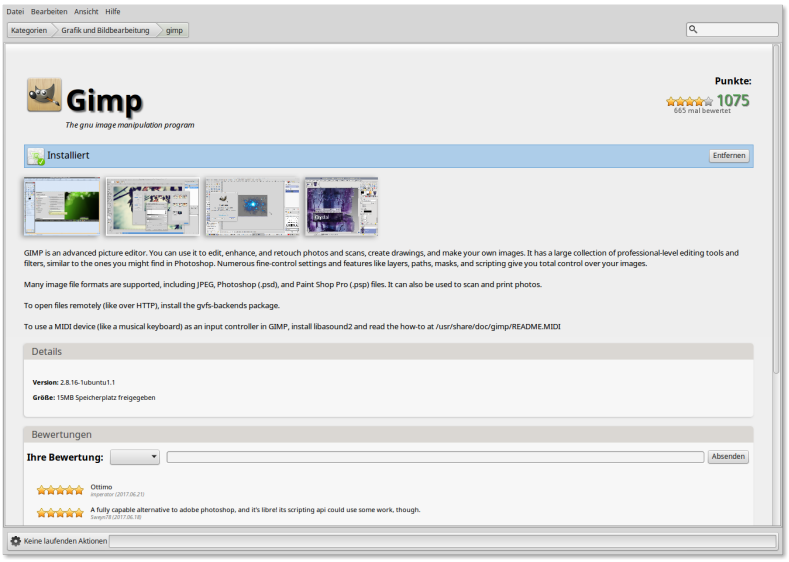
Die graphischen Anwendungsverwaltungs-Programme sind ohne weitere Einarbeitung intuitiv mit der Maus bedienbar:

[4] Fügt man als SuperUser weitere Accounts hinzu, so kann man entscheiden, ob diese sich ebenfalls temporär SuperUser-Rechte verschaffen dürfen.



- Man klickt zunächst auf die jeweilige Programm-Kategorie (beispielsweise Graphik und Bildbearbeitung), um alle entsprechenden verfügbaren Programme aufzulisten.

- Klickt man dann auf gewünschte Programm, so bekommt man eine detailierte Ansicht, ob das Programm installiert ist, welche Funktionen das Programm bietet, und welche Bewertung es durchschnittlich von Benutzern bekommen hat.



Gleich unter der Hauptüberschrift befindet sich eine Status-Zeile, über die das jeweilige Programm mit einem Klick installiert oder wieder entfernt werden kann.

Paketverwaltung mittels `apt`

Die graphischen Verwaltungs-Programme von Debian/Ubuntu/Linux Mint nutzen indirekt die Shell-Anwendung `apt` („Advanced Packaging Tool“) zur Verwaltung und Aktualisierung der installierten Programme und Code-Bibliotheken (siehe auch Ubuntuusers-Wiki).

In einer Shell kann `apt` auch direkt auf einfache Weise genutzt werden. Es gibt dabei folgende Möglichkeiten, das Programm aufzurufen:

- Mit `apt-get update` kann die Liste der verfügbaren Pakete aktualisiert werden. Hierzu prüft `apt`, ob die
- Mit `apt-cache search suchbegriff` ein in Frage kommenden Paket suchen,
- Mit `apt-get install paketname` ein verfügbares Paket installieren.

Zur Vereinfachung ist das Programm `aptitude` empfehlenswert.[5] So lassen sich die beiden Aufruf-Varianten `apt-get` und `"apt-cache"` durch den entsprechenden `aptitude`-Befehl ersetzen:

```
# Programm suchen:
aptitude search suchbegriff

# Programm installieren:
sudo aptitude install paketname
```

Wird versucht mittels `sudo aptitude install paketname` ein Programm zu installieren, das bereits installiert ist, so bleibt der Aufruf ohne Wirkung; mittels `sudo aptitude purge paketname` kann ein Programm wieder deinstalliert werden.

[5] Unter Ubuntu/LinuxMint ist *aptitude* bereits vorinstalliert. Unter Debian kann es mittels `sudo apt-get install aptitude` nachinstalliert werden.

Weitere Infos zur Verwendung von `apt` gibt es im Abschnitt *Administrator-Programme*

Aufbau des Linux-Dateisystems

Die Bezeichnungen für die grundlegenden Verzeichnisse sind auf allen Linux-Systemen (nahezu) identisch:

- Für einen „normalen“ Benutzer ist vor allem das Home-Verzeichnis von Bedeutung, das häufig mit `~` abgekürzt und unter dem Verzeichnis-Pfad `/home/benutzername` abgelegt ist.

  Innerhalb dieses Verzeichnisses kann der Benutzer beliebige weitere Verzeichnisse anlegen und/oder Dateien speichern. Eine Besonderheit unter Linux ist die Kenntlichmachung von Konfigurations-Dateien und -Verzeichnissen: Diese beginnen stets mit einem `.` vor dem eigentlichen Dateinamen; man bezeichnet sie bisweilen auch als „versteckte“ Dateien, da sie beispielsweise bei einem Aufruf der *ls*-Anweisung in einer Shell standardmäßig nicht mit aufgelistet werden. Bei graphischen Datei-Managern wie beispielsweise *caja* können die versteckten Dateien meist mittels einer Tastenkombination ein- oder ausgeblendet werden.

- Werden USB-Sticks, Speicherkarten oder externe Festplatten angeschlossen, so werden diese üblicherweise automatisch über den Verzeichnis-Pfad `/media/benutzername/geraetname` eingehängt (außer man hängt die Geräte manuell mittels *mount* ein); auf dieses Verzeichnis hat der jeweilige Benutzer ebenfalls volle Lese- und Schreibrechte.

Auf die übrigen Verzeichnisse kann meist nur lesend zugegriffen werden.

Zu beachten ist ebenfalls, dass unter Linux zwischen einer Groß- und Kleinschreibung in Datei- und Verzeichnisnamen unterschieden wird (bei Windows ist dies nicht der Fall). Die Dateien `hallo-welt.txt` und `Hallo-Welt.txt` würden somit als zwei verschiedene Dateien angesehen werden. Dies ist zu beachten, wenn man zugleich auch mit Windows-Dateisystemen arbeiten muss: Wäre auf einem solchen bereits eine Datei namens `hallo-welt.txt` vorhanden, und man würde die Datei `Hallo-Welt.txt` dorthin kopieren, so würde dadurch die bestehende Datei überschrieben.

Standard-Verzeichnisse eines Linux-Systems

Zur Organisation der Dateien eines Linux-Systems halten sich die meisten Distributionen an einen bestimmten Standard („Filesystem Hierarchie Standard“). Dieser umfasst die folgenden Verzeichnisse:

- `/` Das Verzeichnis `/` (auch „Wurzelverzeichnis“ genannt) stellt den Basispfad des Systems dar.

- `/bin` In diesem Verzeichnis („binaries“) befinden sich wichtige Programme für Anwender, die immer verfügbar sein müssen, beispielsweise die Shell `bash` oder das Programm `ls` zur Anzeige von Verzeichnis-Inhalten.

- `/boot` In diesem Verzeichnis befinden sich die zum Hochfahren des Systems unbedingt erforderlichen Dateien. Am wichtigsten ist dabei der Kernel, üblicherweise eine Datei mit dem Namen `vmlinuz-versionsname` (andere Namen sind ebenfalls möglich).

- `/dev` In diesem Verzeichnis („devices“) befinden sich ausschließlich Geräte-Dateien. Diese speziellen Dateien stellen eine einfach nutzbare Schnittstelle zur Hardware dar.

  Für jede Festplatte und ihre Partitionen existiert im `/dev/`-Verzeichnis ein eigener Eintrag. Beispielsweise bezeichnet, sofern vorhanden, `/dev/hda` ist die erste IDE-Festplatte, `/dev/sda` die erste SCSI-Festplatte im System. Höhere Buchstaben (`sdb`, `sdc`) stellen weitere Festplatten oder externe Speichermedien dar, Zahlen am Ende (`sda1`, `sda2`) benennen die Partitionen der Festplatten.[1]

- `/etc` In diesem Verzeichnis („etcetera“) befinden sich zahlreiche Konfigurationsdateien, die Einstellungen zu den installierten Programmen sowie grundlegende Systeminformationen enthalten; viele dieser Dateien haben eigene Manpage, die in einer Shell mittels eines Aufrufs der Form `man /etc/fstab` aufgerufen werden kann.

- `/home` In diesem Verzeichnis befinden sich die persönlichen Verzeichnisse der einzelnen Benutzer.

- `/lib` In diesem Verzeichnis („libraries“) befinden sich die wichtigsten Funktionsbibliotheken des Systems; diese Dateien sollten nicht manuell verändert werden.

- `/media` In diesem Verzeichnis werden externe Datenträger als Unterverzeichnisse eingebunden. Bei aktuellen Ubuntu- und LinuxMint-Versionen werden automatisch erkannte USB-Sticks, Speicherkarten, externe Festplatten usw. in ein Verzeichnis der Art `/media/benutzername/name-des-datentraegers` eingebunden.

- `/proc` In diesem Verzeichnis („processes“) sind keine gewöhnlichen Dateien enthalten, sondern Schnittstellen zum Linux-Kernel. Jedes laufende Programm wird hier in einem Unterverzeichnis geführt, dessen Dateien viele Informationen beispielsweise über den aktuellen Programmstatus enthalten.

- `/root` In diesem Verzeichnis befinden sich die (persönlichen) Dateien des Systemverwalters („Root“, „Superuser“). Das Verzeichnis liegt im Wurzelverzeichnis, damit der Systemverwalter auch dann auf seine (Konfigurations-)Dateien zu-

[1] Da auf einer Festplatte nur vier primäre Partitionen möglich sind, wird häufig eine erweiterte Partition angelegt, die den größten Teil der Festplatte umfasst. In der erweiterten Partition können dann „logische Laufwerke“ angelegt werden. Diese erhalten grundsätzlich die Partitionsnummern ab `5`. Enthält eine Festplatte also eine primäre und eine erweiterte Partition, in der sich wiederum zwei logische Laufwerke befinden, gibt es auf dieser Platte die Partitionen `1`, `2`, `5` und `6`. Die primäre Partition ist `1`, die erweiterte ist `2`, und die beiden logischen Laufwerke sind `5` und `6`.

greifen kann, wenn durch einen Fehler der Zugriff auf andere Partitionen nicht mehr möglich ist.[2]

- `/sbin` In diesem Verzeichnis („superuser-binaries“) befinden sich ebenfalls – ähnlich wie im `/bin`-Verzeichnis – wichtige Programme, die allerdings für den Systemverwalter gedacht sind. Sie erfüllen Funktionen, auf die ein normaler Benutzer keinen Zugriff hat.
- `/tmp` Dieses Verzeichnis („temporary“) kann von jedem Benutzer und jedem Programm als temporäre Ablage für Dateien verwendet werden.
- `/usr` In diesem Verzeichnis („unix system ressources“) befinden sich der größte Teil der installierten Software. Auf vielen Systemen befinden sich innerhalb von `/usr` mehr Daten als in allen anderen Dateien zusammen. Die ausführbaren Programmdateien sind meist in `/usr/bin`, Programmbibliotheken in `/usr/lib` abgelegt.

  In Netzwerken, an die viele gleichartige Systeme angeschlossen sind, wird dieses Verzeichnis häufig auf einem zentralen Server gespeichert, und alle anderen Computer greifen über das Netzwerk darauf zu.
- `/var` In diesem Verzeichnis („variables“) befinden sich hauptsächlich Dateien, die sich ständig verändern. Beispielsweise werden hier Log-Dateien gespeichert.
- `/opt` In diesem Verzeichnis („optional“) werden bei Bedarf sehr große Programme gespeichert, die nicht unmittelbar zum System gehören. Bei knappem Festplattenspeicher kann dieses Verzeichnis – wie das `/home`-Verzeichnis – auf einer externen Festplatte oder einer anderen Partition abgelegt werden.

Spezielle Dateien

Der Begriff „Datei“ wird unter Linux sehr weit gefasst; letztendlich stellt jeder Datenaustausch mit einem Speichermedium eine Datei dar. Auch Verzeichnisse sind demnach „Dateien“, ebenso wie die Einhängepunkte der einzelnen Geräte im `/dev`-Verzeichnis.

... to be continued ...

Datei-Rechte

... to be continued ...

[2] Häufig wird bei der Installation eines Linux-Systems für das `/home`-Verzeichnis eine eigene Festplatte verwendet oder eine eigene Partition angelegt, um bei einer möglichen Neuinstallation des Systems die persönlichen Daten unverändert übernehmen zu können.

Hilfreiche GUI-Programme

Im folgenden Abschnitt sind einige Programme mit graphischer Bedienoberfläche („Graphical User Interface“, kurz: GUI) vorgestellt, die bereits installiert sind oder einfach mittels `apt` nachinstalliert werden können.

Datei-Manager

caja

Der Standard-Dateimanager der Mate-Bedienoberfläche heißt `caja`. Er ist bereits vorinstalliert und wird automatisch geöffnet, wenn man beispielsweise über das Startmenü einzelne Orte wie beispielsweise das Home-Verzeichnis anwählt.[1]

Caja wird überwiegend mit der Maus bedient, hat allerdings auch ein paar nützliche Tastenkombinationen: Beispielsweise kann mittels `Ctrl 1`, `Ctrl 2` und `Ctrl 3` zwischen einer Symbol-, Listen- oder Kompakt-Ansicht der einzelnen Verzeichnisse umgeschaltet werden. Die Symbol-Ansicht kann beispielsweise praktisch sein, um sich eine schnelle Übersicht über ein Bilder-Verzeichnis zu verschaffen. In den meisten Fällen ist allerdings die Listen-Ansicht hilfreicher, da hierbei auch die Dateigröße und das Datum der letzten Änderung angezeigt wird; klickt man auf eine dieser Spaltenbezeichnungen, so werden die Dateien gemäß diesen Kriteriums sortiert aufgelistet (bei zweimaligem Anklicken in umgekehrter Reihenfolge.

Über das Menü `Bearbeiten -> Einstellungen` kann unter der Rubrik `Ansichten` festgelegt werden, welche der Ansichten standardmäßig gewählt werden soll („Neue Ordner anzeigen mit“).

gnome-commander

Möchte man nicht nur mit lokalen Verzeichnissen arbeiten, sondern Daten auch auf andere Rechner im Netzwerk/Internet übertragen, so bieten sich Datei-Manager an, die standardmäßig in zwei Fenster-Hälften aufgeteilt sind. Man kann somit beispielsweise in der linken Fensterhälfte ein lokales Verzeichnis ausgewählt haben, und in der rechten Fensterhälfte

[1] Der Standard-Dateimanager der ebenfalls weit verbreiteten Desktop-Umgebung Cinnamon heißt nemo, bei der für alte Hardware optimierten Umgebung LXDE heißt der Standard-Dateimanager PCMan. Diese sind bezüglich des Aussehens und der Bedienung dem Datei-Manager `caja` sehr ähnlich.

ein Verzeichnis eines entfernten Rechners öffnen; als Daten-Übertragungs-Protokolle sind dabei sowohl *SSH* wie auch FTP üblich.

Ein „klassischer“ derartiger Datei-Manager ist der so genannte *Midnight Commander*, der allerdings nur innerhalb eines Shell-Fensters aufgerufen werden kann (was Linux-Einsteiger leider zunächst oft abschreckt). Obwohl ich den Midnight Commander nur allzu sehr empfehlen kann (er ist bei mir neben Vim das wohl meist genutzte Programm), gibt es auch Programme mit einer graphischen Bedienoberfläche, die ähnliche Funktionen bieten – die wohl simpelste Variante ist eben der Gnome Commander. Er kann folgendermaßen installiert werden:

```
sudo aptitude install gnome-commander
```

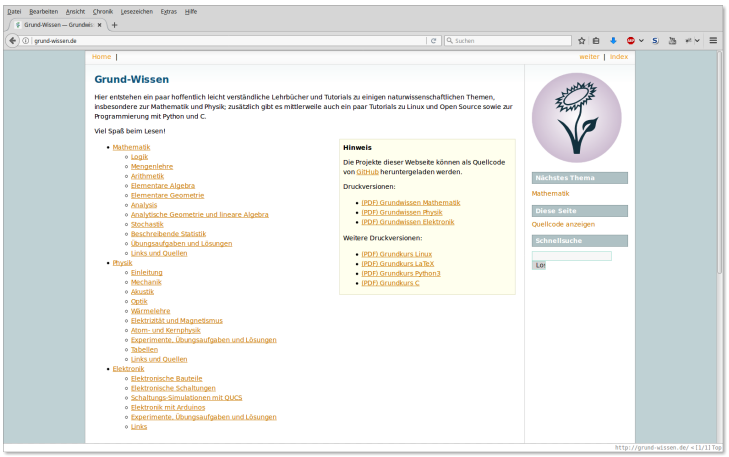
Anschließend kann dieser Datei-Manager aus einer Shell heraus mittels `gnome-commander` oder über das Startmenü (meist Rubrik `Zubehör`) gestartet werden.

Die Navigation innerhalb der einzelnen Fensterhälften kann mittels der Maus oder über die Cursor-Tasten erfolgen, mit der `Tab`-Taste kann zwischen den beiden Fensterhälften gewechselt werden. Weitere nützliche Tasten-Kombinationen sind:

Internet-Anwendungen

`firefox`

Bei firefox handelt es sich um einen Standard-Webbrowser, der bei Debian/Ubuntu/LinuxMint bereits vorinstalliert ist.



Firefox kann daher bei einer neuen Installation unmittelbar genutzt werden; es ist allerdings empfehlenswert, noch einige Plugins nachzuinstallieren, welche den Browser um zusätzliche Features erweitern. Öffnet man über das Menü `Extras -> Add-ons` die Rubrik `Erweiterungen` und gibt im Suchfenster einen Suchbegriff oder den Namen des gewünschten Plugins ein, so werden die entsprechenden Ergebnisse aufgelistet. Die Plugins können dann durch einen simplen Mausklick installiert werden.

Persönlich benutze ich als Plugins den Werbeblocker Adblock Plus, den Skript-Blocker NoScript, den YouTube Video and Audio Downloader, das Entwickler-Tool Firebug sowie die Erweiterung Vimperator für eine *Vim*-artige Bedienung des Browsers.

pidgin

Bei pidgin handelt es um ein Chat-Programm, das mehrere Protokolle unterstützt – unter anderem auch Jabber (XMPP) und IRC. Dank des Zusatz-Plugins `pidgin-otr` ist beim Chatten auch eine verschlüsselte Kommunikation möglich. Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install pidgin pidgin-otr
```

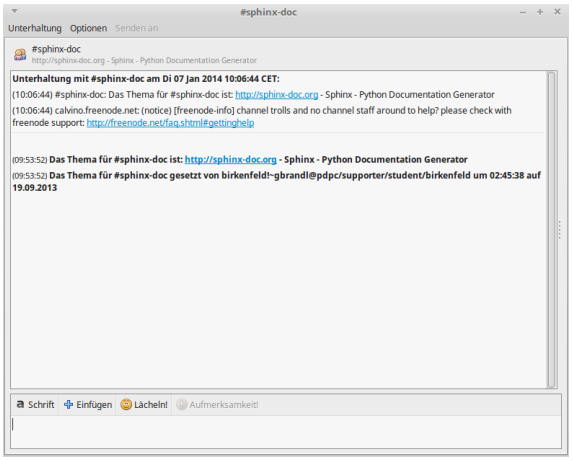
Anschließend kann `pidgin` mittels der gleichnamigen Anweisung über eine Shell oder mittels `Alt F2` gestartet werden; es erscheint zudem ein Eintrag im Programm-Menü unter der Rubrik „Internet“.

Startet man Pidgin, so bekommt man eine Übersicht mit Chat-Kontakten angezeigt; beim erstmaligen Start ist diese Liste leer. Über das Menü `Konten -> Konten verwalten` kann man ein neues Chat-Konto hinzufügen:

- IRC wird für Gruppen-Chats verwendet. Ein neues Konto kann man sich (kostenlos) beispielsweise bei `irc.freenode.net` einrichten, wobei man einen noch nicht bereits vergebenen Benutzernamen und ein Passwort angeben muss; setzt man das Häkchen bei „Passwort speichern“, so muss man den Benutzernamen beziehungsweise das Passwort bei einem erneuten Start von Pidgin nicht erneut eingeben.[2]

  Anschließend kann man über das Menü `Buddys -> Einen Chat betreten` einen Chat-Kanal auswählen. Die meisten Gruppen-Chats sind offen zugänglich, so dass man kein Passwort angegeben muss. Über den Button `Raumliste` bekommt man eine Übersicht über alle „Chat-Rooms“ angezeigt, inklusive der Anzahl der jeweils teilnehmenden Benutzer. Hat man einen Chat ausgewählt (beispielsweise `#vim`), so kann man diesen durch Anklicken des jeweiligen Buttons wahlweise betreten oder diesen mittels `Chat hinzufügen` in die Buddy-Liste aufnehmen. Um einen in der Buddy-Liste gespeicherten IRC-Chat zu betreten, genügt ein Doppelklick auf den jeweiligen Eintrag.

[2] Auch wenn das Passwort für ein Benutzerkonto in Pidgin gespeichert wird, so ist eine zusätzliche Sicherung des Passworts beispielsweise mittels des Passwort-Managers *Keepassx* sinnvoll, falls man beispielsweise den Account einmal mit einem anderen Chat-Programm oder auf einem anderen PC nutzen möchte.



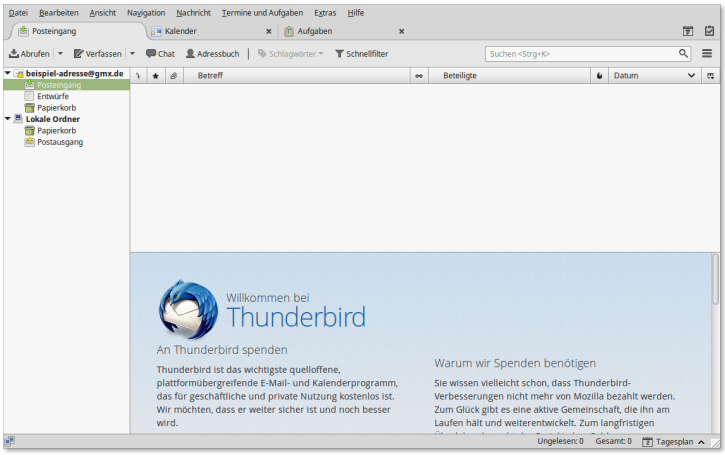
- Jabber (XMPP) wird für Chats mit einzelnen Freunden („Buddies“) verwendet. Für einen solchen Chat brauchen beide Kommunikationspartner ein XMPP-Konto. Ein solches lässt sich beispielsweise (kostenlos) bei `jabber.ccc.de` einrichten, wobei wiederum ein noch nicht existierender Benutzername und ein Passwort angegeben werden muss. Auch hier kann Pidgin die Konten-Daten wieder speichern, so dass sie bei einem Neustart von Pidgin nicht nochmals eingegeben werden müssen.

  Anschließend kann man über das Menü `Buddys -> Buddy hinzufügen` das XMPP-Konto eines Bekannten in die eigene Buddy-Liste speichern; haben sich beide Nutzer gegenseitig in ihrer Buddy-Liste gespeichert, so wird gegenseitig der Status des anderen Benutzers eingeblendet (online, abwesend, offline, oder eine benutzerdefinierte Nachricht). Um mit der jeweiligen Person zu chatten, genügt ein Doppelklick auf den jeweiligen Eintrag in der Buddy-Liste.

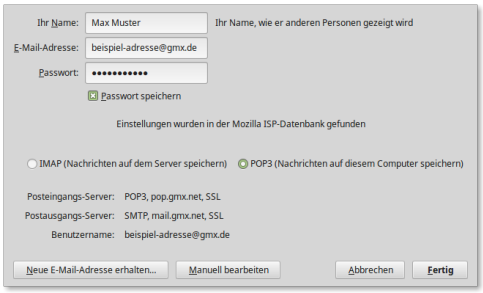
  Haben beide Benutzer (unabhängig davon, welches Chat-Programm sie benutzen) einen OTR-Support installiert, so kann die Kommunikation verschlüsselt werden. In Pidgin kann das OTR-Plugin über das Menü `Werkzeuge -> Plugins` aktiviert werden, indem man in der Liste ein Häkchen bei `Off-the-Record Nachrichtenversand` setzt. In einem Chat-Fenster erscheint dann ein eigener Menü-Eintrag namens `OTR`, über den eine verschlüsselte Unterhaltung begonnen beziehungsweise beendet werden kann.

thunderbird

Bei thunderbird handelt es sich um einen Standard-Email-Cient, der bei Debian/Ubuntu/LinuxMint üblicherweise bereits vorinstalliert ist; ist dies nicht der Fall, so kann er mittels `sudo aptitude install thunderbird` nachinstalliert werden.



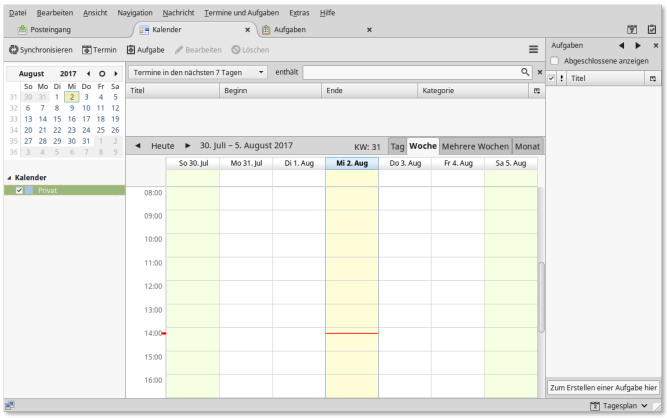
Startet man Thunderbird zum ersten Mal, so erscheint ein Einrichtungs-Assistenz-Fenster, über das ein Email-, RSS-Newsreader- oder auch Chat-Konto eingerichtet werden kann. Möchte man keine neue Email-Adresse einrichten, sondern ein bestehendes Email-Konto (beispielsweise bei www.gmx.de, www.web.de, o.ä) nutzen, so kann man in der Thunderbird-Startseite auf und anschließend auf `Bestehende Email-Adresse verwenden` klicken. Anschließend genügt es, den Benutzer-Namen, die Emailadresse und das Passwort anzugeben, den Rest übernimmt der Einrichtungs-Assistent von alleine.[3]



In der Haupt-Ansicht können die Emails wahlweise nach Eingangsdatum, nach Absender oder nach Betreff sortiert werden; durch Drücken der Tasten `1` bis `5` kann einer Email auch ein Prioritäts-Schlagwort zugewiesen werden, wodurch wichtige Emails automatisch farblich hervorgehoben werden (durch ein nochmaliges Drücken der jeweiligen Ziffer wird die Markierung wieder aufgehoben. Klickt man im Posteinang auf eine Email, so wird deren Inhalt automatisch in der unteren Fensterhälfte angezeigt. Thunderbird bietet von sich aus auch ein Adressbuch zum Verwalten von Kontakt-Emailadressen.

Ähnlich wie bei *Firefox* kann man auch bei Thunderbird durch Add-ons erweitert werden. Klickt man auf das Menü `Extras -> Addons`, so kann man unter der Rubrik `Erweiterungen` gezielt nach verfügbaren Erweiterungen suchen. Eine empfehlenswerte Erweiterung ist `Lightning`: Dieses Add-on erweitert Thunderbird um einen Kalender und Aufgabenplaner.

[3] Je nach Email-Provider muss zunächst über die Online-Bedienoberfläche eine Nutzung der IMAP- beziehungsweise POP3-Funktion vorab freigeschaltet werden. Bei `www.gmx.de` muss man beispielsweise über in der Rubrik `Emails` zunächst auf den `Einstellungen`-Button und anschließend auf `POP3/IMAP-Abruf` klicken; dort kann dann ein Haken bei `E-mails per externem Programm (Outlook, Thunderbird) versenden und empfangen` gesetzt werden. Diese „Umständlichkeit“ liegt nicht an Firefox, sondern am jeweiligen Email-Provider – vermutlich entgehen diesem nämlich Werbe-Einnahmen, da sich der „Kunde“ bei einer Nutzung von Thunderbird kaum noch auf der Webseite des Email-Providers aufhält.



Links:

- Thunderbird-Dokumentation (en.)
- Tasten-Kombinationen in Thunderbird (en.)

Büro-Programme

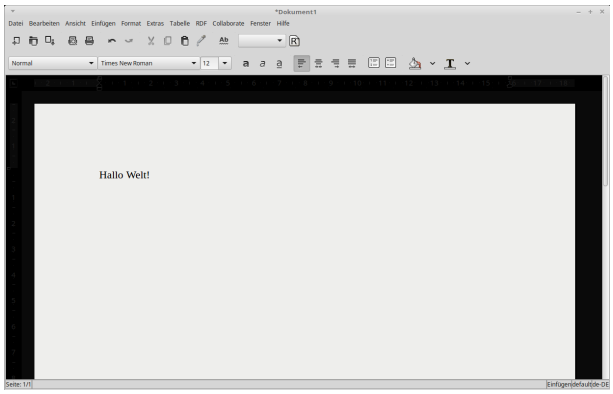
abiword

Bei abiword handelt es sich um eine schlanke Alternative zum standardmäßig installierten Textverarbeitungs-Programm LibreOffice-Writer. Im Vergleich zu letzterem benötigt `abiword` wesentlich weniger Festplattenspeicher, es lädt wesentlich schneller und verursacht im laufenden Betrieb eine geringere CPU- und Arbeitsspeicher-Auslastung.

`abiword` kann folgendermaßen installiert werden:

```
sudo aptitude install abiword
```

Die Bedienung von `abiword` ist einfach und intuitiv; viele Icons und Arbeits-Routinen orientieren sich an LibreOffice beziehungsweise Microsoft Office. Abiword ist zudem mittels in Python oder C++ geschriebenen Plugins erweiterbar.



Nachteilig bei der Verwendung von `abiword` ist lediglich, dass die Formatierung von ursprünglich mit Microsoft Word erstellten Dokumenten teilweise nicht richtig wiedergege-

ben wird. Das Programm bietet also nicht den gesamten Funktionsumfang von LibreOffice, aber ist für ein schnelles Verfassen eigener Dokumente in den meisten Fällen ausreichend.

gedit

Bei gEdit handelt es sich um einen einfachen, dafür allerdings leicht zu bedienenden Text-Editor. Unter Debian/Ubuntu/LinuxMint kann `gedit` folgendermaßen installiert werden:

```
sudo aptitude install gedit gedit-latex-plugin gedit-plugins
```

Anschließend kann der Editor aus einer Shell heraus mittels `gedit` oder über den gleichnamigen Eintrag im Startmenü (üblicherweise unter der Rubrik `Zubehör`) gestartet werden.

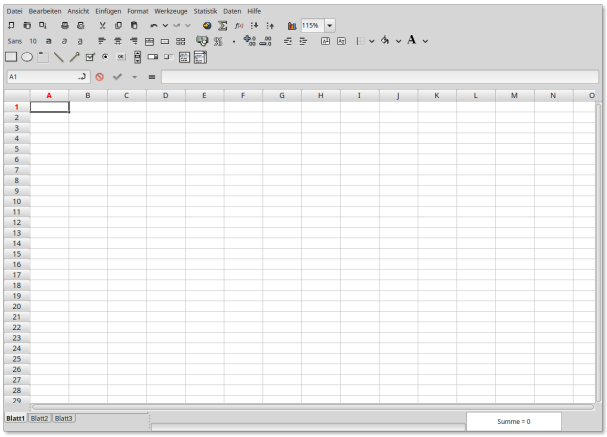
Aktiviert man über das Menü `Bearbeiten -> Einstellungen` die entsprechenden Plugins, so bietet `gedit` unter anderem ein Syntax-Highlighting für verschiedene Datei-Typen, eine Rechtschreibprüfung, sowie die Möglichkeit eigene „Snippets“ zu definieren. Damit sind beliebig umfangreiche „Eingabe-Templates“ gemeint, die durch Eingabe eines zugehörigen Kurzwortes und ein Drücken der `Tab`-Taste in das aktuelle Dokument eingefügt werden können. Da die Snippet-Syntax der von *Vim-Ultisnips* ähnlich ist, ist `gedit` als „einfaches“ Programm für Einsteiger und gelegentliche Nutzer durchaus empfehlenswert; an den Funktionsumfang von *Vim* kommt der Editor allerdings noch lange nicht heran.. ;-)

Links:

- gedit Manual (en.)
- gedit Snippets Howto (en.)

gnumeric

Bei gnumeric handelt es sich um eine schlanke Alternative zum standardmäßig installierten Tabellen-Programm LibreOffice-Calc. Im Vergleich zu letzterem benötigt `gnumeric` wesentlich weniger Festplattenspeicher, es lädt wesentlich schneller und verursacht im laufenden Betrieb eine geringere CPU- und Arbeitsspeicher-Auslastung.



`gnumeric` kann folgendermaßen installiert werden:

```
sudo aptitude install gnumeric gnumeric-plugins-extra
```

Die Bedienung von `abiword` ist einfach und intuitiv. Viele Icons und Arbeits-Routinen orientieren sich an LibreOffice beziehungsweise Microsoft Office; da das Programm bezüglich des Grundumfangs jedoch eher ‚schlicht' gehalten ist, findet man die ‚wichtigen' Funktionen bei nur gelegentlicher Nutzung meist schneller.

Gnumeric kann zudem mit benutzerdefinierten, in Python geschriebenen Funktionen erweitert werden; hierzu muss man über das Menü `Werkzeuge -> Plugins` und im erscheinenden Auswahlfenster einen Haken bei `Python-Funktionen` setzen; anschließend kann beispielsweise über das Menü `Werkzeuge -> Python-Eingabefenster` aus Gnumeric heraus ein Python-Interpreter gestartet werden. Gibt man dort `import Gnumeric` ein, so wird das zu Gnumeric gehörende Python-Modul geladen (das allerdings nur verfügbar ist, wenn Python aus Gnumeric heraus gestartet wird).

... to be continued ...

Links:

- The Gnumeric Manual (en.)
- Programming Gnumeric using Python (en.)

libreoffice

Die Libreoffice-Suite ist ein mit Microsoft Office vergleichbares Programm-Paket. Es umfasst folgende Teilprogramme:

- `Libreoffice Calc` ist ein mit Microsoft Excel vergleichbares Tabellen-Kalkulations-Programm.
- `Libreoffice Writer` ist ein mit Microsoft Word vergleichbares Programm zum Schreiben von Briefen oder anderen Texten.
- `Libreoffice Impress` ist ein mit Microsoft PowerPoint vergleichbares Programm zum Erstellen von Präsentationen.
- `Libreoffice Base` ist ein mit Microsoft Access vergleichbares Programm zum Verwalten von Datenbanken.

Die übrigen Programme `Libreoffice Draw` zum Erstellen von Zeichnungen sowie `Libreoffice Math` zum Setzen von mathematischen Formeln sind bei mir bislang noch nie zum Einsatz gekommen; anstelle `Draw` verwende ich wesentlich lieber *Inkscape*, und zum Setzen von Texten (insbesondere mit mathematischen Formeln) nutze ich bevorzugt LaTeX.

zathura

Bei zathura handelt es sich um einen schlanken und schnellen PDF-Betrachter; er kann mit einer *Vim* -artigen Steuerung bedient werden.

```
sudo aptitude install zathura
```

Zathura kann wahlweise auch mit einem dunklen Farbschema genutzt werden, das in der Datei `~/.config/zathura/zathurarc` festgelegt wird.

Zathura kann weitgehend mit der Tastatur bedient werden. Dabei gibt es folgende hilfreiche Hotkeys:

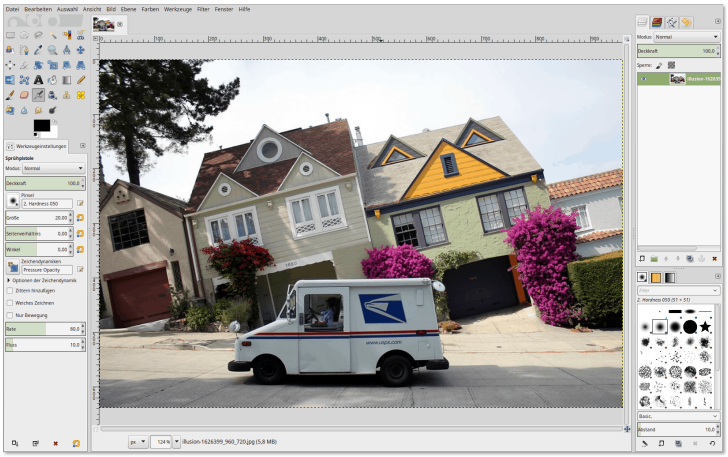
Taste	Bedeutung
`Tab`	Inhaltsverzeichnis anzeigen (sofern vorhanden)
`PageDown`, `PageUp`	Seitenweise vor-/zurückblättern
`J`, `K`	Seitenweise vor-/zurückblättern
`j`, `k`	Zeilenweise vor-/zurückblättern
`/`, `?`	Suche nach Text (vor- beziehungsweise rückwärts
`n`, `N`	Nächstes/Vorheriges Ergebnis der Textsuche
`f`	Links auf der aktuellen Seite anzeigen; mit `Zahl Enter` gelangt man dorthin
`gg`, `G`	Erste beziehungsweise Letzte Seite öffnen
`Zahl G`	Seite mit angegebener Seitenzahl öffnen
`+`, `-`	PDF-Datei vergrößert/verkleinert darstellen
`a`	PDF-Datei auf Bildschirmhöhe einpassen
`s`	PDF-Datei auf Bildschirmbreite einpassen
`d`	Zwei Seiten nebeneinander darstellen (Buchmodus)
`r`	Seite um 90 Grad nach rechts drehen
`Ctrl r`	Zwischen „Light“- und „Dark“-Modus wechseln

Eine PDF-Datei kann zudem mit `R` neu geladen werden; dies ist allerdings kaum nötig, denn Zathura liest die Datei automatisch neu, wenn sie verändert wurde.

Bildbearbeitungs-Programme

gimp

Bei gimp handelt es sich um ein Programm zur Bearbeitung von PNG-, JPG- und anderen Bilddateien. Gimp gehört zum Standard von Debian/Ubuntu/LinuxMint und ist somit üblicherweise bereits vorinstalliert; falls dies nicht der Fall ist, so kann es mittels `sudo aptitude install gimp` nachinstalliert werden.



Beim ersten Start von Gimp über `Menü -> Grafik -> GIMP` oder durch Eingabe von `gimp` in einer Shell sollte zunächst über das Menü `Fenster` der so genannte Einzelfenster-Modus aktiviert werden; in diesem seit der Programm-Version 2.8 verfügbaren Modus verhält sich das Programm bezüglich des Fenster-Managements ebenso wie andere Graphik-Programme (beispielsweise *Inkscape* für SVG-Graphiken).

Gimp eignet sich gut zum Zuschneiden von Bildern, zum Anpassen der Farbsättigung und des Kontrasts, sowie zum Retuschieren von Photos bezüglich roter Augen, o.ä. Gimp kann auch als Malprogramm verwendet werden, allerdings empfiehlt sich hierfür dann auch die Verwendung eines Graphik-Tablets.

Links:

- Gimp Scripting with Python (en.)

inkscape

Bei inkscape handelt es sich um ein Programm zur Erstellung von Vektor-Graphiken („Scaleable Vector Graphics“, SVG). Dieses Programm wird im eigenen Abschnitt *Inkscape* näher beschrieben.

Multimedia-Programme

audacious

Bei audacious handelt es sich um einen ebenso kleinen wie vielseitigen Audio-Player. Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install audacious
```

Gestartet werden kann `audacious` anschließend über `Startmenü -> Multimedia -> audacious` oder über die gleichnamige Shell-Anweisung.

Audacious bietet mehrere „Themes“, die das Aussehen der Bedienoberfläche maßgeblich beeinflussen:

- Mit dem als Standard eingestellten „GTK-Interface“ sieht `audacious` fast aus wie ein Browser, wobei die einzelnen Tabs den einzelnen Wiedergabelisten entsprechen; die obere Zeile beinhaltet gewöhnliche Icons zur Wiedergabe, Lautstärke-Regulierung, Zufallswiedergabe, sowie zum Öffnen von Dateien.

  Durch einen Klick auf das `a`-Icon links oben im Fenster kann man über den Menü-Eintrag `Datei -> Einstellungen` das Erscheinungsbild und weitere Eigenschaften anpassen.

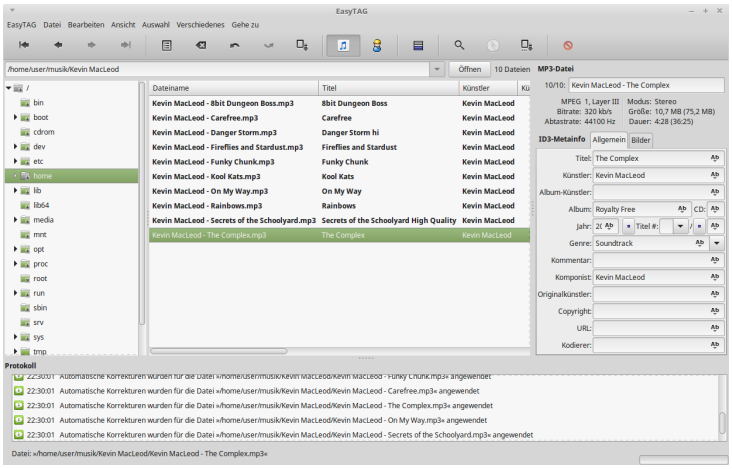
- Mit dem „Winamp-Interface“ erscheint `audacious` mit einer Bedien-Oberfläche mit minimaler Größe; die aktuelle Playliste kann optional durch einen Klick auf den `Pl`-Button eingeblendet werden.

Die Wahl des Interfaces ist Geschmacksache: Das GTK-Interface bietet eine übersichtliche und mit der Maus gut bedienbare Darstellung (optional mit zusätzlichen Informationen), das Winamp-Interface hingegen bietet auf sehr kleinen (Touch-)Displays Vorteile.[4]

easytag

Bei easytag handelt es sich um ein Programm zur schnellen und einfachen Bearbeitung von Audio-Metadaten („ID3-Tags“).

```
sudo aptitude install easytag
```



Das Programm bietet, wie in der obigen Abbildung zu sehen, Eingabe-Formulare für die einzelnen ID3-Tag-Felder einer Audio-Datei; man kann im mittleren Auswahl-Fenster auch mehrere Dateien auswählen, um beispielsweise in allen Dateien auf einmal das Künstler- oder Album-Feld auszufüllen.

Easytag bietet bei Bedarf über `Ansicht -> Scanner anzeigen` eine einfache Möglichkeit, ID3-Tags automatisch anhand des jeweiligen Datei- beziehungsweise Verzeichnisnamens auszufüllen.

[4] Persönlich verwende ich einen Raspberry Pi mit einem auf die GPIO-Pins aufgesteckten 3,5”-Display und einer externen USB-Soundkarte als portable Wiedergabe-Station für aktive Lautsprecherboxen.

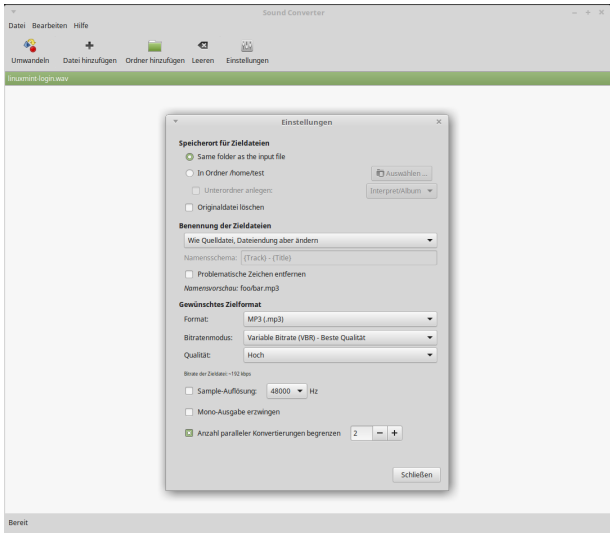
soundconverter

Bei soundconverter handelt es sich um ein Programm, das Audiodateien in beliebige andere Audio-Formate umwandeln kann. Das Programm unterstützt neben MP3 auch OGG, WAC, FLAC, MPC; es können wahlweise einzelne Dateien oder auch ganze Verzeichnisse umgewandelt werden.

Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install soundconverter lame gstreamer1.0-plugins-ugly
```

Das letzte Paket ist nur erforderlich, sofern auch MP3-Dateien konvertiert werden sollen.



Die Bedienung ist denkbar einfach:

- Man klickt zunächst in der Symbolleiste auf den „Einstellungen“-Button einen Zielpfad für die neuen Audiodateien (Standard: Das gleiche Verzeichnis wie die Original-Dateien) sowie das gewünschte Ausgabe-Format aus.
- Anschließend klickt man auf „Datei hinzufügen“ beziehungsweise „Ordner hinzufügen“, wodurch die zu konvertierenden Dateien im Hauptfenster aufgelistet werden.
- Durch einen Klick auf „Umwandeln“ wird die Konvertierung gestartet. Der Fortschritt wird dabei anhand eines „Ladebalkens“ links neben der jeweiligen Datei angezeigt.

Der Soundconverter kann übrigens mit der gleichen Vorgehensweise genutzt werden, um Audio-Spuren aus beispielsweise `.flv`- oder `.mp4`-Videos zu extrahieren.

vlc

Bei vlc handelt es sich um einen weithin bekannten und vielseitigen Audio- und Videoplayer, der eine Vielzahl an gängigen Dateitypen unterstützt (MPG, AVI, FLV, MP3, OGG, usw). Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install vlc vlc-nox
```

Das Paket `vlc-nox` ist nicht zwingend nötig, bringt allerdings den Vorteil mit sich, dass damit der VLC-Player in einer Shell mittels `nvlc Datei` auch im Text-Modus starten lässt.

VLC lässt sich auch zum Abspielen von DVDs und CDs nutzen. Während unter neueren Versionen von Ubuntu/LinuxMint bereits alle dafür nötigen Codecs vorinstalliert sind, müssen auf Debian-Systemen folgende Pakete manuell nachinstalliert werden:

```
sudo aptitude install libc6 w32codecs libdvdcss2
```

Der VLC-Player lässt sich intuitiv über mittels der graphischen Bedienoberfläche bedienen; zudem gibt es folgende bisweilen nützliche Tastenkürzel:

Taste	Bedeutung
`Leertaste`	Pause/Fortsetzen
`Ctrl` ↑, `Ctrl` ↓	Lautstärke erhöhen beziehungsweise verringern
`Alt` →, `Alt` ←	Video um 10 s vor- beziehungsweise zurückspulen
`Ctrl` →, `Ctrl` ←	Video um 1 min vor- beziehungsweise zurückspulen
`-`, `+`, `=`	Video langsamer, schneller beziehungsweise mit normaler Geschwindigkeit abspielen

Obwohl der VLC-Player auch Audio-Dateien abspielen kann, nutze ich ihn persönlich fast ausschließlich zum Abspielen von Video-Dateien; für Audio-Dateien nutze ich hingegen bevorzugt *audacious*.

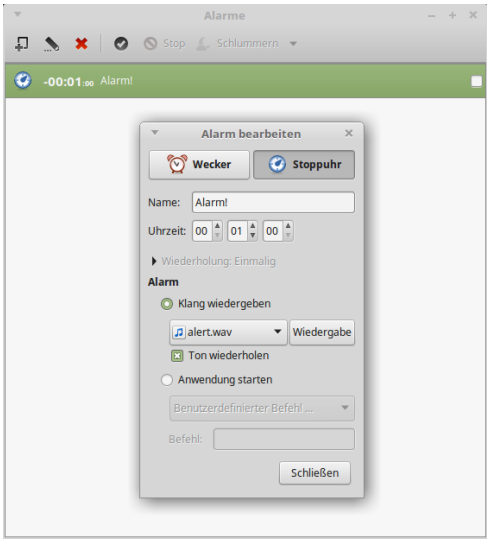
Hilfsprogramme

alarm-clock-applet

Bei alarm-clock-applet handelt es sich um einen schlichten Erinnerungsdienst mit Stoppuhr- und Uhrzeit-Funktion. Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install alarm-clock-applet
```

Anschließend kann `alarm-clock-applet` mittels der gleichnamigen Anweisung über eine Shell oder mittels `Alt F2` gestartet werden; zusätzlich wird das Programm automatisch in die Liste der beim Start automatisch geladenen Programme aufgenommen, was über `Startmenü -> Steuerzentrale -> Persönlich -> Startprogramme` geändert werden kann. Ist das Programm aktiv, so erscheint in im Symbol-Abschnitt der Taskleiste ein kleines Wecker-Symbol.



Das Programm ist sehr einfach bedienbar: Klickt man mit einem Doppelklick auf das Wecker-Icon, so werden die Alarme angezeigt. Durch einen Klick auf das entsprechende Icon (oben links) wird ein neuer Alarm definiert, wobei man zwischen der Wecker- und der Stoppuhr-Funktion wählen kann:

- Bei der Wecker-Funktion wird die angegebene Zeit (Stunden: Minuten: Sekunden) als Uhrzeit interpretiert.
- Bei der Stoppuhr-Funktion wird die angegebene Zeit (Stunden: Minuten: Sekunden) als Countdown-Timer interpretiert.

Zusätzlich ist es empfehlenswert, eine Audio-Datei anzugeben, die abgespielt werden soll, wenn der Alarm ausgelöst wird; persönlich finde ich die standardmäßig bereits vorhandene Datei `/usr/share/sounds/purple/alert.wav` zwar nicht besonders klangvoll, aber durchaus als Wecker-Signal geeignet.[5] Lässt man das Feld frei, so wird beim Auslösen des Alarms nur ein Hinweis-Fenster am oberen rechten Rand des Bildschirms geöffnet.

Nach dem neuen Erstellen eines Alarms wird dieser automatisch aktiviert und in die Alarm-Liste aufgenommen; dort kann er ausgewählt und über die Icons in der Symbolleiste wahlweise bearbeitet, gestoppt oder auch wieder gelöscht werden. Nach dem Auslösen beziehungsweise Stoppen eines Alarms bleibt dieser dennoch in der Liste erhalten und kann somit als Vorlage für ein neues Alarm-Event verwendet werden.

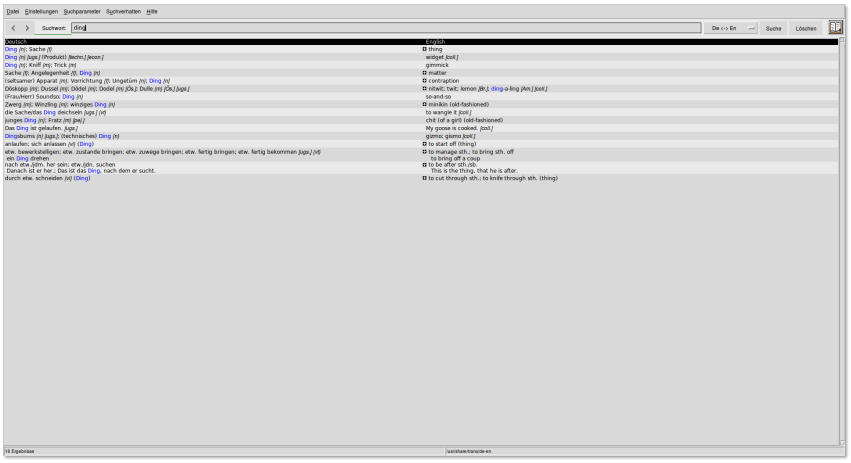
ding

Bei ding handelt es sich um ein ein deutsch-englisch-sprachiges Wörterbuch. Das Programm und das zugehörige Wörterbuch kann folgendermaßen installiert werden:

```
sudo aptitude install ding trans-de-en
```

Anschließend kann `ding` mittels der gleichnamigen Anweisung über eine Shell oder mittels `Alt F2` gestartet werden; es erscheint zudem (zumindest bei Debian/Ubuntu/LinuxMint mit Mate-Desktop) ein Eintrag im Programm-Menü unter der Rubrik „Büro“.

[5] Nach der Datei `alert.wav` kann in einer Shell auch manuell mittels `locate alert.wav` gesucht werden.

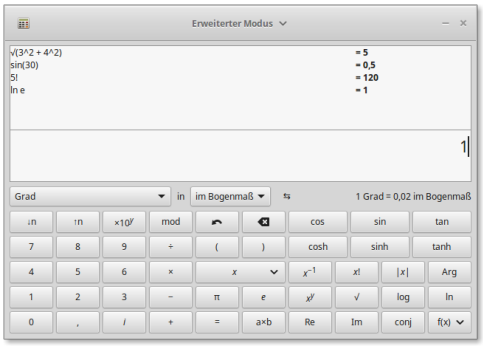


Das Wörterbuch ist intuitiv bedienbar: Gibt man in der Eingabezeile einen Begriff ein und drückt `Enter`, so werden die entsprechenden Ergebnisse angezeigt. Gibt man dann erneut Text ein, so wird der Inhalt der Eingabezeile in der Regel automatisch überschrieben; notfalls kann diese mittels `Ctrl U` auch ohne ein Markieren des Textes mittels der Maus gelöscht werden.

Klickt man auf die Pfeiltasten neben dem Eingabefeld, so können die vorherigen Übersetzungen wieder angezeigt werden.

gnome-calculator

Bei gnome-calculator handelt es sich um ein unter Debian/Ubuntu/LinuxMint standardmäßig bereits installiertes Taschenrechner-Programm; es kann aus einer Shell heraus mittels `gnome-calculator` oder über das Startmenü aufgerufen werden (meist ist es in der Rubrik `Zubehör` zu finden).



Die Bedienung des Programms kann wahlweise über die Tastatur oder mittels der Maus erfolgen; bleibt man mit dem Maus-Cursor eine Weile über den einzelnen Buttons, so bekommt man eine kurze Beschreibung der jeweiligen Funktionen als Hilfe eingeblendet. Drückt man die `Enter`-Taste, so wird der eingegebene Term ausgewertet und das Ergebnis sowohl im oberen Fensterabschnitt (der Eingabe-History) sowie als Vorlage für weitere Rechnungen in der Eingabezeile angezeigt; auf diese Weise kann schnell mit den (Teil-)Ergebnissen weiter gerechnet werden. Drückt man hingegen die `Esc`-Taste, so wird der Inhalt der Eingabe-Zeile gelöscht.

guake

Bei guake handelt es sich um eine Shell, die auf Tastendruck am oberen Bildschirmrand ein- und ausgeblendet werden kann.

`guake` kann folgendermaßen installiert werden:

```
sudo aptitude install guake
```

Anschließend kann `guake` mittels der gleichnamigen Anweisung aus einer Shell heraus oder mittels `Alt F2` gestartet werden. Ist `guake` aktiv, so kann mit den Standard-Einstellungen durch Drücken der Taste `F12` ein Shell-Fenster am oberen Rand des Bildschirms eingeblendet werden; dieses kann wie jedes gewöhnliche Shell-Fenster genutzt werden. Klickt man allerdings mit der rechten Maustaste in dieses Fenster, so können die über den Menü-Eintrag `Einstellungen` unter anderem die Tastenkombinationen für `guake` geändert werden.

Das wichtigste Tastenkürzel betrifft das An- und Ausblenden des `guake`-Fensters; ich habe hierfür die Taste `F1` ausgewählt, da diese bei Debian/Ubuntu/LinuxMint gewöhnlich Hilfe-Seiten zum aktuellen Programm öffnet. Dies kann auch störend sein, wenn man gelegentlich aus Versehen auf diese Taste drückt. Wird die Taste hingegen von `guake` belegt, so wird die ursprüngliche Funktion dieser Taste dadurch überlagert.

Mit `guake` können, ähnlich wie bei einem Webbrowser, auch mehrere Shell-Sitzungen als Tabs nebeneinander geöffnet sein. Ich nutze dafür allerdings lieber *tmux*, so dass bei mir die entsprechende von `guake` bereit gestellte Funktion ungenutzt bleibt.

keepassx

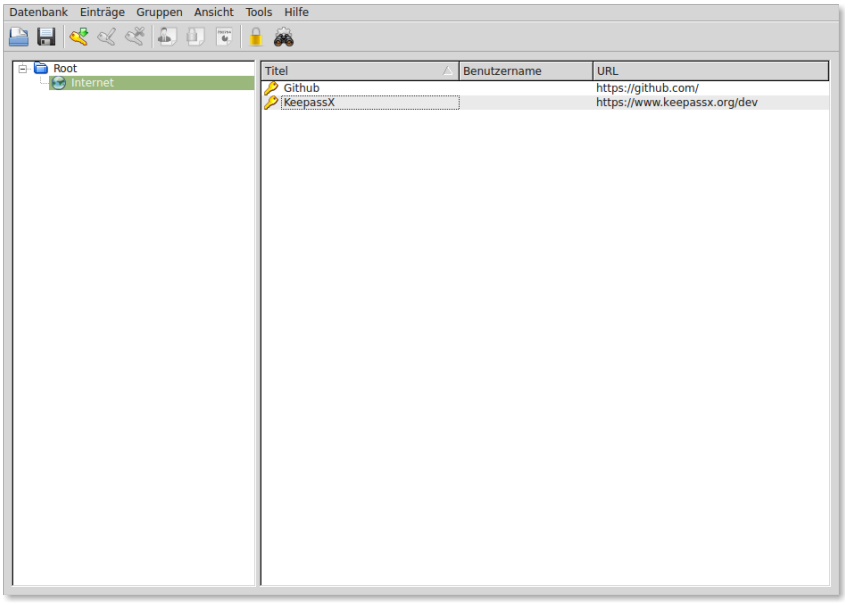
Bei keepassx handelt es sich um ein Programm zur sicheren Verwaltung von Passwörtern. Der Grundgedanke hierbei ist, dass man sich künftig nur *ein* gutes Passwort (oder den Namen einer selbst gewählten Schlüsseldatei) merken muss und dafür alle weitere Passwörter in verschlüsselter Form zentral verwaltet werden.

KeepassX kann folgendermaßen installiert werden:

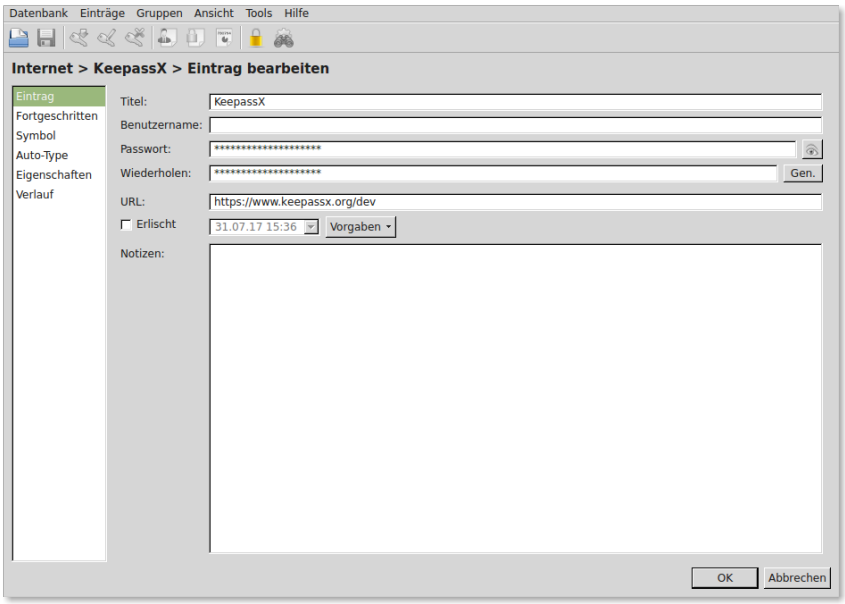
```
sudo aptitude install keepassx
```

Anschließend kann `keepassx` mittels der gleichnamigen Anweisung über eine Shell oder mittels `Alt F2` gestartet werden; es erscheint zudem ein Eintrag im Programm-Menü unter der Rubrik „Sonstiges“.

Beim ersten Start von KeepassX wird man unmittelbar durch ein Eingabe-Formular aufgefordert eine neue Datenbank für Passwörter anzulegen; dazu muss man angeben, in welcher Datei beziehungsweise in welchem Pfad die Datenbank gespeichert werden soll. Zudem muss wahlweise ein Passwort für diese Datenbank angegeben werden oder eine beliebige Datei aus dem Dateisystem ausgewählt werden, die als „Schlüsseldatei“ fungieren soll. Wählt man die letztere Variante, so muss zum Freischalten der Datenbank kein Passwort angegeben werden, sondern stattdessen die korrekte Datei ausgewählt werden.



Einträge in die Passwort-Datenbank können zur besseren Übersichtlichkeit in verschiedene Gruppen unterteilt werden; diese werden im linken Fenster-Abschnitt von KeepassX angezeigt.



Wer `keepassx` häufiger nutzt, wird sich vermutlich freuen, dass es für einige Funktionen auch Tastenkombinationen gibt: Den im aktuell ausgewählten Eintrag hinterlegten Benutzernamen kann man beispielsweise mit `Ctrl b` in die Zwischenablage kopieren, das zugehörige Passwort mit `Ctrl c`. Nach einer bestimmten Zeit, die über das Menü `Tools -> Einstellungen` festgelegt werden kann, wird die Zwischenablage automatisch wieder gelöscht, damit das Passwort nicht aus Versehen an anderer Stelle im Klartext eingefügt wird.

redshift

Bei redshift handelt es sich um ein Programm, das den Bildschirm je nach Tageszeit und geographischer Lage rötlich einfärbt – dies ist für die Augen definitiv entspannend.[6] Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install gtk-redshift
```

Anschließend kann Redshift über `Programm-Menü -> Steuerzentrale -> Persönlich -> Startprogramme` in die Liste der beim Start automatisch aufgerufenen Programme aufgenommen werden. Hierzu erstellt man einen neuen Eintrag, wobei der Programmaufruf folgende Syntax aufweisen sollte:

```
# Allgemeine Syntax:
gtk-redshift -l breitengrad:längengrad

# Beispiel für Augsburg (50 Grad Nord, 10 Grad Ost):
gtk-redshift -l 50:-10
```

Den Breiten- und Längengrad eines Ortes kann man beispielsweise über Wikipedia-Einträge ausfindig machen; alternativ kann der Ort auf OpenRouteService gesucht werden; dort wird der Breiten- und Längengrad automatisch oben rechts eingeblendet.

Mit den Ortsangaben kann das Programm beispielsweise errechnen, zu welchen Tageszeiten die Sonne am höchsten steht oder die Dämmerung einsetzt. Tagsüber wird der Bildschirm kaum eingefärbt, in den dunklen Stunden wird der Blau-Anteil reduziert beziehungsweise der Rot-Anteil erhöht.

Möchte man, beispielsweise bei der Gestaltung eines Flyers, gelegentlich auch die „echten“ Farben angezeigt bekommen, so kann `redshift` jederzeit über das Icon im Symbolfeld der Taskleiste an- beziehungsweise ausgeschaltet werden.

wine

Auch wenn es unter Linux für die meisten Zwecke eigene, auf dem Open-Source-Prinzip basierende Programme gibt, lassen sich bei Bedarf – allerdings ohne Garantie – kommerzielle Windows-Programme auch mittels des Windows-Emulators Wine installieren beziehungsweise bedienen. Um Wine unter Ubuntu/LinuxMint zu installieren, sollte man folgendermaßen vorgehen:

```
sudo add-apt-repository ppa:ubuntu-wine/ppa
sudo apt-get update
sudo apt-get install wine1.8
```

Nach der Installation von Wine können Windows-Programme (auch Installations-Programme) mittels `wine programm.exe` aufgerufen werden. Mittels `winecfg` bezie-

[6] Der physikalische Grund für diesen Effekt ist im Abschnitt Die Farbwahrnehmung des Menschen beziehungsweise den dortigen Unterabschnitt Tag- und Nachtsehen näher beschrieben.

hungsweise des entsprechenden Eintrags im Programm-Menü kann eine graphische Konfigurations-Oberfläche gestartet werden.

Das Emulieren von Programmen mit Wine hat allerdings Nachteile: Einerseits kann es (insbesondere bei alten Geräten) zu einer erheblichen Prozessor- und Speicherlast führen, andererseits sind Windows-Programme in der Regel nicht auf Linux-Systeme abgestimmt und stellen letztlich Sicherheitslücken dar. Wine sollte daher nur dann genutzt werden, wenn es unbedingt erforderlich ist, man dem Software-Hersteller vertrauen kann und (noch) kein entsprechendes Linux-Programm existiert.

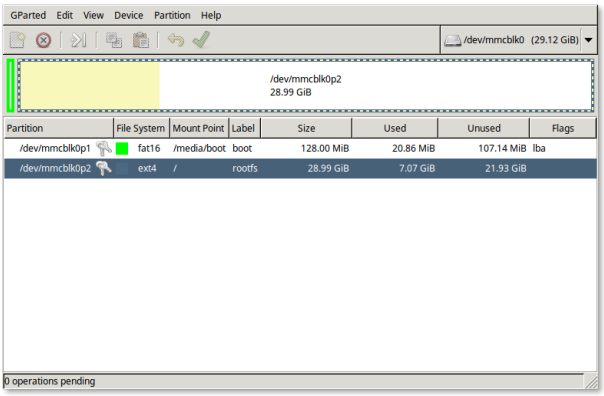
Systemverwaltungs-Programme

gparted

Bei gparted handelt es sich um einen Partitions-Manager, mit dem verschiedene Partitionen auf lokalen Festplatten oder USB-Sticks erstellt und verwaltet werden können.Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install gparted
```

Gestartet werden kann `gparted` nur mit SuperUser-Rechten, da das Programm bei nicht gewollter Benutzung auch die Partitionierung der System-Festplatte unbrauchbar machen kann. Aus einer Shell heraus kann `gparted` also wahlweise mit `sudo gparted` oder mit `sudo gparted geraetname` gestartet werden. Im letzteren Fall kann nur das entsprechende Gerät bearbeitet werden, im ersteren (beispielsweise `/dev/sdb`), im ersteren kann das zu bearbeitende Speichermedium über den Auswahl-Button oben rechts beziehungsweise das erscheinende Menü festgelegt werden.



Gibt es auf einem Speichermedium noch keine Partition, so kann eine solche über das `Partitionen`-Menü neu erstellt werden; als Standard sollte eine `GPT`-Partitionierung verwendet werden.

Über das Menü `Partition` können dann bestehende Partitionen gelöscht sowie neue hinzugefügt werden. Als Dateisysteme sind dabei folgende üblich:

- Linux-Partitionen sollten mit dem Dateisystem `ext4` formatiert werden. Dieses ist ausgesprochen verlässlich und unterstützt auch Dateigrößen von mehr als `4 GB`.

- USB-Sticks und andere Speichermedien, die auch auf auf Windows-Rechnern, Kopiergeräten usw. eingesetzt werden sollen, sollten mit `fat32` formatiert werden. Dieses Dateisystem unterstützt allerdings keine *Symlinks* und unterscheidet nicht zwischen Groß- und Kleinschreibung.
- Partitionen, die als „erweiterter Arbeitsspeicher“ genutzt werden sollen, müssen mit dem Dateisystem `swap` formatiert werden.

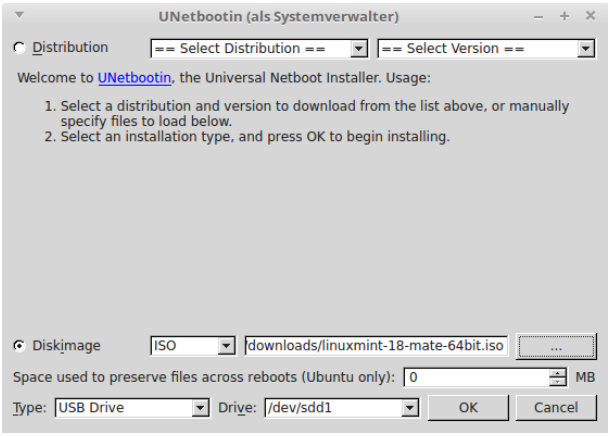
Über das Menü `Partition -> Manage Flags` können zudem auch „Markierungen“ für einzelne Partitionen gesetzt werden; beispielsweise muss bei USB-Sticks, auf denen ein Linux-Live-System installiert ist, die Option `bootable` gesetzt werden, damit ein Booten von diesem Gerät ermöglicht wird.

unetbootin

Bei unetbootin handelt es sich um ein Programm zum Erstellen von bootfähigen Live-USB-Sticks. Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install unetbootin
```

Gestartet werden kann `unetbootin` nur mit SuperUser-Rechten, da das Programm – wie jedes Partitionierungs-Programm – bei nicht gewollter Benutzung auch die Partitionierung der System-Festplatte unbrauchbar machen kann.



Die Bedienung von `unetbootin` ist prinzipiell einfach: Nachdem man eine ISO-Datei der gewünschten *Linux-Distribution* heruntergeladen hat, wählt man dieses unter der Rubrik `Diskimage` aus, indem man auf den `...`-Button klickt. Am unteren Rand des Fensters muss dann noch der Device-Name des Ziel-USB-Sticks angegeben werden (diesen kann man beispielsweise durch Eingabe von *lsblk* in einem Shell-Fenster ermitteln). Anschließend muss man nur noch auf `Ok` klicken, die Einrichtung des Live-USB-Sticks nimmt Unetbootin dann von selbst vor.

Weiterführende Tutorials

Erstellen von 2D-Graphiken mit `inkscape`

Inkscape ist ein umfangreicher Vektorgraphik-Editor, der für Windows wie Linux gleichermaßen als Open-Source-Tool frei verfügbar ist.

Unter Debian/Ubuntu/LinuxMint lässt sich Inkscape mittels des gleichnamigen Paketes installieren:

```
sudo aptitude install inkscape fonts-mathjax fonts-mathjax-extras
```

Inkscape bietet bereits in der Grundversion den Import und Export der gängigsten Graphik-Formate, darunter auch PNG und PDF.

LaTeX-Formeln einbinden

Möchte man auch LaTeX-Formeln in seine Zeichnungen integrieren, empfiehlt sich die `tex text`-Erweiterung. Sie kann von der Homepage des Entwicklers heruntergeladen werden. Zur Installation muss lediglich das Archiv entpackt und der Inhalt in den Ordner `~/.config/inkscape/extensions` verschoben oder kopiert werden.

```
# In das Erweiterungs-Verzeichnis wechseln
cd ~/.config/inkscape/extensions

# Datei herunterladen
wget https://pav.iki.fi/software/textext/textext-0.4.4.tar.gz

# Datei entpacken
tar xvzf textext-0.4.4.tar.gz
```

Die Erweiterung benötigt mit `pdf2svg` und `pstoedit` zwei weitere Pakete, die – falls sie nicht verfügbar sind – über die Paketverwaltung installierbar sind:

```
sudo aptitude install pdf2svg pstoedit
```

Anschließend kann die Erweiterung in der Menüzeile über `Erweiterungen -> Tex Text` aufgerufen werden.

Zusätzlich kann mittels den „Mathjax“-Schriften auch normaler Text in einem LaTeX-ähnlichen Stil gesetzt werden, um ein einheitliches Schriftbild im Dokument zu erreichen.

Einheiten des Lineals und des Gitters anpassen

Die beiden Lineal-Leisten am linken und oberen Rand des Hauptfensters zeigen Längen standardmäßig in Pixeln an. Um dies zu ändern, muss man in den Dokument-

Eigenschaften (Tastenkürzel `Ctrl D`) unter der Rubrik „Seite“ als Standard-Einheiten `cm` (oder `mm`) einstellen.

Die Einstellungen des Gitters können im Menü über `Datei -> Inkscape-Einstellungen` unter der Rubrik „Gitter“ geändert werden. Ich selbst verwende folgende Einstellungen:

Bei diesen Einstellungen werden die Gitterlinien (Tastenkürzel `#`) bei Vollansicht des gesamten Dokuments (bei mir meist DinA4, Tastenkürzel `5`) in `cm`-Rasterung angezeigt; zoomt man näher in einen Bereich hinein, wechselt die Rasterung auf `mm`.

Multilayer-SVG-Dateien als PDF exportieren

Leider unterstützt SVG (noch) keine Multipage-Dokumente. Man kann sich jedoch damit behelfen, indem man für jede neue Seite eine neue Ebene („Layer“) erstellt. Hierfür kann man das Menü `Ebene` nutzen, oder mit `Ctrl L` das Ebenen-Werkzeug am rechten Bildschirmrand einblenden.

Während in PDF-Dateien die Seiten von vorne nach hinten nummeriert werden (in Anzeigeprogrammen bzw. Inhaltsverzeichnissen von oben nach unten), so werden in Inkscape die einzelnen Ebenen von unten nach oben nummeriert. Die zuletzt hinzugefügte „Schicht“ ist somit die oberste. Man kann die einzelnen Ebenen im Ebenen-Werkzeug leicht nach unten und oben verschieben, sollte aber auf die passende Reihenfolge achten, wenn man die einzelnen Schichten als jeweils neue Seiten einer PDF-Datei exportieren möchte.

Ein solcher Export einer mit Inkscape erstellten Multilayer-SVG-Datei in eine Multipage-PDF-Datei ist dank des Skripts svglayers2pdfpages.sh von Christoph Haag möglich: Gibt man im Verzeichnis der zu exportierenden SVG-Datei `svglayers2pdfpages.sh svgfile.svg` ein, so wird im gleichen Verzeichnis eine gleichnamige PDF-Datei erzeugt.

PNG-Dateien aus Shell heraus exportieren

Inkscape ermöglicht einen PNG-Export einer SVG-Datei auch aus einer Shell heraus. Beim Aufruf von `inkscape` in einer Shell können hierbei folgende zusätzliche Optionen genutzt werden:

- `-z`: Diese Option verhindert eine Starten der graphischen Bedienoberfläche.
- `-d wert`: Mit dieser Option wird die Auflösung der PNG-Zieldatei festgelegt; der Standardwert liegt bei `96 dpi`.
- `-D`: Diese Option legt fest, dass nur der Zeichenbereich der SVG-Datei, nicht das gesamte Dokument exportiert werden soll. Dies ist empfehlenswert, wenn in Inkscape das Standard-DinA4-Format als Dokumenteinstellung belassen wird, die erstellte Graphik jedoch deutlich kleiner ist (ohne diese Option bekäme eine solche Abbildung einen entsprechend „großen“ Rahmen).
- `-e dateiname`: Gibt den Dateinamen der PNG-Zieldatei an.

Zum Erstellen der Abbildungen auf meiner Grund-Wissen-Seite nutze ich folgendes Mini-Shell-Skript, um mir aus allen SVG-Dateien eines Verzeichnisses die zugehörigen PNG-Dateien zu erstellen:

```
for i in *.svg; do inkscape $i -z -d 150 -D -e $(basename $i .svg).png; done
```

Hierfür habe ich mir, um dies nicht wiederholt eingeben zu müssen, ein eigenes Alias in der Konfigurationsdatei `~/.zshrc` erstellt:

```
INK1='for i in *.svg; do inkscape $i -z -d 150 -D -e $(basename $i .svg).png;␣
↪done'
```

Somit genügt es, im jeweiligen Ordner in einer `INK1` aufzurufen, um alle darin enthaltenen SVG-Dateien als PNG zu exportieren.

SVG-Dateigrößen mit `svgo` optimieren

Inkscape speichert erstellten SVG-Dateien standardmäßig als Inkscape-SVG-Dateien, die unter anderem Informationen darüber enthalten, welche Gitter-Optionen gesetzt wurden, welche Auswahl-Optionen aktiv sind, wo sich Führungslinien befinden, welche Optionsfenster beim Schließen geöffnet waren usw. Dies ist zwar bei der Bearbeitung angenehm, macht die resultierenden SVG-Dateien aber erheblich größer als nötig.

Um schlanke SVG-Dateien beispielsweise auf einer Homepage verlinken zu können, sollte ein SVG-Optimierer verwendet werden. Persönlich verwende ich inzwischen svgo, ein in Node.js geschriebenes Programm für die Kommandozeile.

Unter Ubuntu / Linux Mint lässt sich `svgo` folgendermaßen installieren:

```
sudo aptitude install npm nodejs-legacy

git clone https://github.com/svg/svgo
```

```
cd svgo

sudo npm install -g svgo
```

Anschließend kann `svgo` mittels folgender Syntax genutzt werden:

```
# Eine Einzelne Dateien optimieren:
svgo -i originaldatei.svg -o neue-datei.svg

# Verzeichnis mit SVG-Dateien optimieren:
# (Erst Sicherheitskopie erstellen!!!)
svgo -f pfadname
```

Persönlich erstelle ich für jedes Projekt die einzelnem Inkscape-Dateien in einem strukturell identisch aufgebauten `Original`-Verzeichnis, und kopiere von dort die SVG-Dateien ins Zielverzeichnis. Im Zielverzeichnis wiederum, in dem die Dateien optimiert werden sollen, rufe ich anschließend `svgo -f .` auf. Dadurch werden die Inkscape-SVGs meist um 60-80% kleiner – teilweise liegt der Kompressionsgrad sogar bei über 90%.. :-)

Links

- Inkscape-Projektseite

Am besten lernt man Inkscape – wie so oft – mittels „Learning by doing“. Begleitend sind dabei beispielsweise folgende Tutorials hilfreich:

- Inkscape-Wikibook
- Inkscape-Einführung der Universität Göttingen
- Offizielles Inkscape-Wiki
- Inkscape-Crashcourse
- Guide to a Vector Drawing Program (en.)
- Quick Guide to Inkscape (en.)

Einfache bis komplexere Beispiele finden sich als zusätzliche Anregungen auf folgenden Seiten:

- Guide to a vector drawing program (en.)
- 35 Tutorials to create vector graphics (en.)
- Drawing Gears (en.)

Erstellen von 3D-Modellen mit `freecad`

FreeCAD ist, wie der Name schon sagt, ein freies Programm zum Erstellen von dreidimensionalen Modellen („Computer Animated Design“). Mit FreeCAD lassen sich beispielsweise

relativ einfach mechanische Modelle oder Architekturmodelle mit genauen Abmessungen erstellen.

Einen guten Einblick in die Funktionen von FreeCAD gibt beispielsweise die deutschsprachige FreeCAD-Videoreihe auf Youtube.

Installation

FreeCAD kann unter Ubuntu über folgendes Paket installiert werden:

```
sudo aptitude install freecad
```

Möchte man allerdings stets auch an den neuesten Entwicklungen teilhaben, so ist es empfehlenswert, die Paket-Quellen des Entwickler-Teams mit aufzunehmen:

```
sudo add-apt-repository ppa:freecad-maintainers/freecad-daily
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install freecad-daily freecad-daily-doc
```

Freecad kann anschließend über das Start-Menü oder aus einer Shell heraus mittels `freecad` für die Standard-Version beziehungsweise `freecad-daily` für die aktuellste Entwickler-Version gestartet werden.

Arbeitsflächen („Werkbänke“)

Beim ersten Programmstart von FreeCAD erscheint ein fast leerer Bildschirm. Dies liegt daran, dass noch keine „Werkbank“ ausgewählt ist; als solche werden in FreeCAD für bestimmte Zwecke zusammengestellte Icon-Leisten bezeichnet. Wählt man im mittleren Auswahlmenü beispielsweise die Arbeitsfläche/Werkbank `Sketcher`, so werden die für das Erstellen und Bemaßen von 2D-Skizzen nützlichen Werkzeuge als anklickbare Icons eingeblendet.[1]

Folgende Arbeitsflächen in FreeCAD mit der Grundversion verfügbar:

- Der Arbeitsbereich `Start` dient als Einstiegshilfe in das Programm und bietet einige Tips sowie Links zu Tutorials (beispielsweise Youtube oder das FreeCAD-Handbuch).
- Der Arbeitsbereich `Sketcher` ist für das Erstellen von Skizzen mit Rand- bzw. Zwangsbedingungen ausgelegt.
- Der Arbeitsbereich `PartDesign` bietet Werkzeuge für die Erstellung von 3D-Körpern aus 2D-Skizzen sowie die Weiterbearbeitung von 3D-Körpern mittels zusätzlicher Skizzen.
- Der Arbeitsbereich `Part` wird zur Erstellung von geometrischen Grundkörpern verwendet; zudem können beliebige geometrische Körper weiter bearbeitet werden.

[1] Welche Arbeitsfläche standardmäßig ausgewählt sein soll, wenn FreeCAD gestartet wird, kann über das Menü `Bearbeiten -> Einstellungen` festgelegt werden.

- Der Arbeitsbereich `Complete` beinhaltet alle Werkzeuge der anderen Arbeitsbereiche. Die Symbolleiste ist dadurch stark gefüllt; die einzelnen Symbol-Gruppen können jedoch nach eigenen Vorstellungen platziert werden, indem man je eine Gruppe mit gedrückter linker Maustaste an eine andere Stelle in der Symbolleiste bewegt.

Zu Beginn sind die wohl wichtigsten Arbeitsflächen die PartDesign- und die Part-Werkbank, die in den folgenden Abschnitten vorgestellt werden. Weitere Arbeitsflächen können als Addons hinzugefügt werden.

Fenster-Ansichten

In FreeCAD ist die aktuelle 3D-Ansicht im Hauptfenster mittig platziert. Da mehrere Dateien gleichzeitig geöffnet werden können, ist zur Übersicht und für ein einfaches Wechseln der Fenster unmittelbar unter dem Hauptfenster eine entsprechende Tab-Leiste angebracht.

Links davon ist üblicherweise eine so genannte „Combo-Ansicht“ eingeblendet:

- Der obere Teil der Combo-Ansicht enthält im oberen Teil eine Art Inhaltsverzeichnis („Modell-Baum“) des aktuellen Projekts, im dem einzelne Objekte ausgewählt werden können (mit gedrückter `Ctrl`-Taste auch mehrere auf einmal). Zudem können in diesem Fenster auch Bauteile gruppiert sowie durch Drücken der Leertaste aus beziehungsweise wieder eingeblendet werden können.

  Der obere Teil der Combo-Box hat zusätzlich eine zweite anwählbare Rubrik namens „Aufgaben“. Sofern eine Bearbeits-Funktion es erfordert, wechselt die Combo-Box automatisch in diese Rubrik und ermöglicht es die einzelnen Funktions-Parameter einzustellen.

- Der untere Teil der Combo-Ansicht enthält ebenfalls zwei Rubriken mit jeweils editierbaren Tabellen. In der Rubrik „Ansicht“ kann das Aussehen eines Objekts (Liniendicke, Selektierbarkeit usw.) festgelegt werden, in der Rubrik „Daten“ kann die Positionierung sowie die Größe eines Objekts durch Zahlenwerte genau festgelegt werden.

Über dem Hauptfenster sind eine oder mehrere Symbol-Leisten zu finden, wobei je nach ausgewählter Werkbank die verschiedene Icon-Gruppen angezeigt werden.

Unter dem Hauptfenster befindet sich standardmäßig ein Ausgabefenster (unter anderem für Fehler und Warnungen) sowie eine Python-Konsole; diese beiden Fenster können anfangs guten Gewissens geschlossen werden, um auf dem Bildschirm mehr Platz für die 3D-Ansicht zu bekommen. Bei Bedarf können die geschlossenen Fenster wieder über das Menü `Ansicht -> Ansichten -> Ausgabefenster` beziehungsweise `Ansicht -> Ansichten -> Python-Konsole` wieder eingeblendet werden.

Navigation

In FreeCAD können viele Arbeitsschritte mit der Maus vorgenommen werden. Standardmäßig haben die einzelnen Tasten folgende Bedeutung:

- `Linke Maustaste`: Objekt auswählen. Mit gedrückter `Ctrl`-Taste können durch einen Klick auf die linke Maustaste weitere Objekte zur Auswahl hinzugefügt werden; durch einen Klick ins Leere wird die aktuelle Auswahl aufgehoben.

  Mit den Standard-Einstellungen leuchten auswählbare Punkte, Kanten und Flächen in FreeCAD automatisch gelb auf, sobald sich der Maus-Cursor über ihnen befindet. Wählt man eine solche Kante oder Fläche durch einen Klick mit der linken Maustaste aus, so wird sie automatisch grün eingefärbt.

- `Rechte Maustaste`: Durch einen Rechtsklick wird im 3D-Fenster ein Menü mit verschiedenen Darstellungs- und Bearbeitsoptionen geöffnet.

- `Mausrad`: Dreht man am Mausrad, so wird die Ansicht im 3D-Fenster vergrößert beziehungsweise verkleinert („Zoom“). Hält man die mittlere Maustaste gedrückt, so kann die Ansicht verschoben werden („Pan“).

  Hält man das Mausrad gedrückt *und* drückt dann wahlweise die linke oder rechte Maustaste, so kann man die Ansicht um den Maus-Cursor als Drehzentrum drehen, bis man die linke beziehungsweise rechte Maustaste wieder los lässt.

Ist man von einem ähnlichen Programm eine andere Belegung der Maustasten gewöhnt, so kann man im Menü `Bearbeiten -> Einstellungen` unter der Rubrik `Anzeige` auch andere vordefinierte Tastenbelegungen auswählen: Neben der standardmäßigen „CAD-Navigation“ ist auch eine Navigation wie in den Programmen „Blender3D“ oder „Inventor“ sowie speziell für Touchpads ausgelegte Navigation wählbar. Klickt man nach der Auswahl auf das daneben platzierte `Maus`-Icon, so wird die Tasten-Belegung für den jeweiligen Stil eingeblendet.

Die Maus-Navigation kann man leicht ausprobieren, indem man zur `Part`-Werkbank wechselt und oben links in der Symbolleiste auf das Würfel-Icon klickt; hierdurch wird ein neuer Würfel mit Standard-Maßen im Ursprung des Koordinatensystems eingefügt.

Zusätzlich zur Maus-Navigation kann auch das NumPad der Tastatur zur Navigation genutzt werden:

- Mit der Taste `Numpad-0` wechselt die Ansicht in eine schräge („axometrische“) 3D-Ansicht von vorne.

- Mit der Taste `Numpad-1` wird in eine Frontal-Ansicht des Objekts gewechselt ($x-z$-Ebene). Die Taste `Numpad-4` liefert die zugehörige Ansicht von hinten.

- Mit der Taste `Numpad-2` wird in die Vogelperspektive gewechselt ($x-y$-Ebene). Die Taste `Numpad-5` liefert die zugehörige Ansicht von unten.

- Mit der Taste `Numpad-3` wird in eine Seiten-Ansicht des Objekts gewechselt ($y-z$-Ebene, rechte Seite). Die Taste `Numpad-6` liefert die zugehörige Ansicht von der linken Seite.

Klickt man mit der rechten Maustaste in das 3D-Fenster und wählt im erscheinenden Menü `Einpassen` aus, so wechselt die aktuelle Ansicht in eine axometrische Ansicht, die alle in der geöffneten Datei existierenden Objekte beinhaltet.

Zusätzlich kann die Darstellungsweise der Objekte im 3D-Fenster geändert werden, indem man mit der rechten Maustaste in das 3D-Fenster klickt und im Menü auf `Zeichenstil` klickt. Wechselt man in diesem Untermenü beispielsweise auf „Drahtgitter“, so werden die

Objekte nur noch anhand ihrer Konturen dargestellt, jedoch ohne feste Flächen (kann beispielsweise beim Erstellen von zusätzlichen Skizzen auf bestehenden Objekten hilfreich sein).

Für die obigen Navigations-Funktionen existieren jeweils auch Icons, die standardmäßig in der obersten Symbol-Leiste ganz links zu finden sind.

Einstellungen

FreeCAD bietet eine Vielzahl an Anpassungsmöglichkeiten, sowohl was die Bedienung als auch das Aussehen der Arbeitsflächen anbelangt. Die Einstellungen können allgemein über das Menü `Bearbeiten -> Einstellungen` vorgenommen werden. Gewöhnungsbedürftig, aber auch nicht unpraktisch ist es, dass hierbei Einstellungsmöglichkeiten nur für diejenigen Arbeitsbereiche eingeblendet werden, die in der aktuellen Sitzung bereits einmal durch eine entsprechende Auswahl in der Haupt-Symbolleiste aktiviert wurden.

Für einen besseren Kontrast ist es beispielsweise empfehlenswert, kurz in den PartDesign-Arbeitsbereich zu wechseln und dann unter `Bearbeiten -> Einstellungen -> Anzeige` in der Rubrik `Skizze` die Kantenfarbe von weiß auf schwarz umzuschalten.

Zudem ist es empfehlenswert, in der Rubrik `Part Design` in der Rubrik `Allgemein -> Modelleinstellungen` alle drei Häkchen bei „Modell automatisch nach Boolescher Operation überprüfen“, „Modell automatisch nach Boolescher Operation verfeinern“ und „Verfeinere Modell nach Skizzenoperation automatisch“ zu setzen.

Bauteil mit der PartDesign-Werkbank erstellen

Um ein neues Bauteil zu erstellen, sollte zunächst eine zweidimensionale Skizze des Grundrisses erstellt werden. Klickt man bei aktiver PartDesign-Werkbank auf das Skizze-Symbol in der Symbolleiste, so werden die entsprechenden Funktionen des Sketchers aktiviert.[2]

Bei einer neuen Skizze wählt man zunächst aus, für welche Ebene die Skizze gedacht ist. Die Auswahl lässt sich zwar nachträglich jederzeit korrigieren, da das entstehende Bauteil gedreht werden kann, es ist jedoch beispielsweise für die Erstellung einer Bodenplatte durchaus hilfreich, die zugehörige Skizze von vornherein waagrecht in die $x - y$-Ebene zu legen.

Zeichenelemente in Skizze aufnehmen

Um Zeichenelemente in die Skizze aufzunehmen, genügt ein Klick auf das entsprechende Icon in der Symbolleiste; bewegt man den Maus-Cursor über die einzelnen Symbole, so wird am unteren Bildschirmrand eine Kurzbeschreibung der jeweiligen Funktion eingeblendet.

[2] Die Skizze kann wahlweise auch mittels der `Sketcher`-Werkbank erstellt und erst anschließend in der `PartDesign`-Werkbank weiter bearbeitet werden.
Viele Elemente der `Sketcher`-Werkbank sind allerdings auch auf der `PartDesign`-Arbeitsfläche verfügbar, so dass oftmals bereits diese zum Erstellen von Skizzen ausreichend ist.

Nachdem die Formen in der Skizze festgelegt sind, kann mit der Ausrichtung und Dimensionierung der einzelnen Elemente begonnen werden. Hierin unterscheidet sich FreeCAD deutlich von manuell erstellten Skizzen: Die Zeichnung muss nicht von Anfang an mit Präzision angefertigt werden; Objekte können auch im Nachhinein ausgerichtet und mit Zusatz-Bedingungen versehen werden.

Längen und Symmetrien festlegen

FreeCAD vergibt in der Skizze automatisch, sofern möglich, Beschränkungen („Constraints“). Beispielsweise erhalten beim Zeichnen eines Rechtecks die einzelnen Seiten automatisch Vertikal- beziehungsweise Horizontal-Beschschränkungen.

Hat man das ein Rechteck fertig gezeichnet, so ist dieses in seiner Bemaßung und Platzierung noch nicht eindeutig festgelegt; es hat noch so genannte „Freiheitsgrade“. Legt man zwei Werte für Länge und Breite sowie einen Abstand zur x- sowie zur y-Achse fest, so ist das Rechteck hingegen eindeutig festgelegt. Hat eine Zeichnung keine Freiheitsgrade mehr, so spricht man von einer „vollständig eingeschränkten Skizze“; in diesem Fall wird die komplette Skizze grün hervorgehoben.

Mehr Infos zu den Funktionen des Sketchers finden sich im FreeCAD-Manual.

Skizze zu einem 3D-Objekt „aufpolstern“

Ist man mit dem Erstellen der Skizze fertig, so kann man anschließend die Skizze mittels der entsprechenden Funktion in der Symbolleiste „aufpolstern“; so entsteht aus einer zweidimensionalen Skizze ein drei-dimensionales Objekt.

Wählt man bei einem Objekt durch einen Klick mit der linken Maustaste eine einzelne Fläche aus, so kann durch ein erneutes Anklicken des Skizze-Symbols in der Symbolleiste eine neue Skizze auf dieser Fläche erzeugt werden. FreeCAD dreht dabei automatisch die Ansicht so, dass die ausgewählte Fläche von oben als Unterlage der Skizze betrachtet wird. Hat man wiederum eine Skizze fertig erstellt, so sind zweierlei Funktionen möglich:

- Mittels der „Taschen“-Funktion in der Symbolleiste kann man eine Bohrung beziehungsweise Ausfräsung durch das Objekt bewirken; hierbei kann die Bohr- beziehungsweise Frästiefe entweder durch Bemaßung oder über logische Beziehungen (beispielsweise „Bis zur nächsten Oberfläche“ oder „Durch alles“) angegeben werden.
- Mittels der „Aufpolstern“-Funktion in der Symbolleiste kann man aus der Skizze einen neuen 3D-Körper erstellen, der dann senkrecht zur bestehenden Fläche ausgerichtet ist. Auf diese Weise lassen sich auch komplexe Geometrien schichtweise aufbauen.

Eine Besonderheit der „Aufpolstern“-Funktion des PartDesign-Arbeitsbereics liegt darin, dass als Resultat stets ein einzelnes Objekt entsteht. Im Modellbaum entsteht somit eine Hierarchie der Bearbeitungsschritte: Die Grundkörper sind oben angeordnet, die später hinzukommenden Aufpolsterungen und/oder Taschen werden jeweils unten an-

gefügt. Im Modellbaum bleiben die Original-Objekte also erhalten, während sie in der 3D-Hauptansicht von FreeCAD automatisch ausgeblendet werden.[3]

Farben festlegen

Um ein Objekt einzufärben, klickt man im Modellbaum (oberer Teil der Combo-Box) mit der rechten Maustaste auf das gewünschte Objekt und wählt im erscheinenden Menü den Eintrag „Darstellung“ aus. Die Combo-Box wird hierdurch zu einem Eingabe-Formular für die Kanten- und Flächenfarben des Objekts.

Weitere Infos zur PartDesign-Werkbank finden sich im FreeCAD-Manual.

Bauteile mit der Part-Werkbank erstellen

In der `Part`-Werkbank können einerseits dreidimensionale geometrische Grundkörper wie Würfel, Kugel, Zylinder oder Kegel durch ein einfaches Anklicken des jeweiligen Icons in der Symbolleiste hinzugefügt werden; andererseits können bereits bestehende 3D-Objekte beispielsweise mittels logischen Operatoren oder speziellen Funktionen weiter bearbeitet werden.

Einen neuen 3D-Grundkörper kann man durch Anklicken des jeweiligen Icons in der Symbolleiste erstellen. Das neue Objekt wird dabei mit einstellbaren Standard-Maßen am Koordinaten-Ursprung eingefügt. Da das neue Objekt zudem automatisch ausgewählt wird, können die Standard-Maße bei Bedarf sogleich in der unteren Hälfte der Combo-Box abgeändert werden.

Neben den Längen-Maßen kann in der unteren Hälfte der Combo-Box auch die Platzierung eines Objekts geändert werden. Hat man ein Objekt ausgewählt, so erscheint als erster Eintrag im Datenfeld der Combo-Box die Eigenschaft „Placement“. Klickt man auf das Icon mit den drei Punkten in der gleichen Zeile, so ändert sich die Combo-Box in ein Bearbeitungsfenster für exakt angegebene Positionierungen:

- Unter der Rubrik „Verschiebung“ kann vorgegeben werden, um wie weit das ausgewählte Objekt in die x, y beziehungsweise z Achse verschoben werden soll.
- Unter der Rubrik „Drehpunkt“ kann, wie der Name schon sagt, ein Punkt als Drehzentrum festgelegt werden. Dies ist allerdings nur von Bedeutung, sofern die Drehung um einen einzelnen Punkt erfolgen soll.

[3] Dies ist auch der Grund, weshalb es in FreeCAD keine einfache Möglichkeit gibt, die Objekte im Modell-Baum neu anzuordnen: Bei Objekten, die mit Hilfe des PartDesign-Arbeitsbereichs erstellt wurden, könnte so die Objekt-Historie verloren gehen, was zu unerwarteten Ergebnissen führen könnte.

Als Workaround ist es für eine Neuordnung des Modellbaums allerdings möglich, im Anschluss an eine Sicherheitskopie eine neue Gruppe im Modellbaum zu erstellen (Rechtsklick im oberen Teil der Combo-Box, „Gruppe erstellen“ anklicken) und die gewünschten Objekte in der Soll-Reihenfolge per Drag-and-Drop in diese Gruppe zu ziehen. Löscht man die Gruppe anschließend wieder, so bleibt die Reihenfolge der Objekte in der Gruppe erhalten.

Macht man dies mit Objekten aus dem PartDesign-Arbeitsbereich, so muss allerdings geprüft werden, ob dadurch Fehler entstanden sind; notfalls muss man die vorherigen Arbeitsschritte rückgängig machen.

Soll eine Drehung um eine Achse erfolgen, so können die Drehpunkt-Koordinaten beim Wert Null belassen werden; dafür kann in der unteren Hälfte Teil des Bearbeitungsfensters eine Achse ausgewählt sowie ein Drehwinkel festgelegt werden.

In welche Richtung die einzelnen Koordinaten bei der aktuellen 3D-Ansicht zeigen, kann man an dem kleinen Achsenkreuz erkennen, das rechts unten im Hauptfenster eingeblendet ist.

Setzt man zudem einen Haken bei der Option „Änderungen an Objektplacement inkrementell übernehmen“, so kann man schrittweise Änderungen, beispielsweise Verschiebungen entlang einer Achse vornehmen, und bekommt diese jeweils nach einem Klick auf „Anwenden“ unmittelbar angezeigt. Ohne diese Option muss die gesamte Transformation als ein einziger Prozess-Schritt vorgenommen werden.

Boolesche Operationen

Im Part-Arbeitsbereich werden nur diejenigen Icons für boolesche Operationen farbig angezeigt, die auf die aktuelle Auswahl angewendet werden können. Beispielsweise kann die Operation „Schnittmenge“ nur dann angewendet werden, wenn (mindestens) zwei Objekte ausgewählt sind. Dabei muss unter Umständen auf die Auswahl-Reihenfolge geachtet werden: Bei der booleschen Operation „Differenz“ wird beispielsweise das als zweites ausgewählte Objekt aus dem zuerst ausgewählten Objekt ausgeschnitten.

... to be continued ...

Links

- FreeCAD Projektseite (en.)
- FreeCAD Wiki (en.)
-
- A FreeCAD Manual (en., auch PDF)
- FreeCAD-Videoreihe auf Youtube (de.)
- FreeCAD 0.17 Release Notes

Shell-Basics

Die Begriffe „Shell“, „Kommandozeile“ und „Terminal“ sind unter Linux gleichbedeutend für ein Eingabefenster, in dem Programme mit verschiedenen Aufruf-Optionen direkt gestartet werden können.

Etliche text-basierte Programme bieten darüber hinaus die Möglichkeit, aus einer Shell eine eigene Arbeitsumgebung werden zu lassen und/oder zahlreiche Aufgaben des Alltags schnell und einfach zu erledigen.

Syntax-Regeln

Die Shell ist ein vielfältiger Interpreter, um (beliebige) Programme zu starten und deren Rückgabewerte anzuzeigen bzw. weiter zu verarbeiten.

In vielen Fällen müssen einem Programm weitere Informationen übergeben werden, damit es seine Aufgabe in gewünschter Weise erfüllen kann. Es gibt dabei zwei Arten von Informationen, die man Programmen mitteilen kann: Optionen und Argumente. Dabei werden die Optionen immer vor den Argumenten angegeben, so dass die grundlegende Syntax aller Linux-Anweisunge folgende Form hat:

```
programm [-Optionen] [Argumente]
```

Die eckigen Klammern sollen andeuten, dass Optionen und Argumente nicht bei jedem Programmaufruf notwendig sind. Ihre Angabe hängt vom Zweck des Programmaufrufs und den möglichen Parametern eines Programms ab.

Optionen

Optionen können das Verhalten einer Anweisung beeinflussen. Jede Option wird gewöhnlich durch einen einzelnen Buchstaben bezeichnet und beginnt mit einem vorangestellten Minus (`-`).

Beispiel:

- Die Anweisung `ls` zeigt den Inhalt eines Verzeichnisses an, indem es die Namen der enthaltenen Unterverzeichnisse und Dateien auflistet. Will man allerdings nicht einfach nur die Namen der Dateien wissen, sondern auch Zusatzinformationen über Dateigröße, Erstellungsdatum oder ähnliches mitgeteilt bekommen, so muss der Aufruf von `ls` um entsprechende Optionen ergänzt werden:

```
user@linux $ ls -l
```

Die Option `-l` („long“) bewirkt eine ausführlichere Ausgabe, das Verhalten des Programms hat sich durch die Verwendung der Option verändert.

Optionen können miteinander kombiniert werden, indem man weitere Zeichen einfach hinzufügt; beispielsweise gibt `ls -lh` die Dateiliste in ausführlicher Form *und* mit angenehm lesbaren Dateigrößen an.[1] Das Minuszeichen muss also nur ein einziges Mal verwendet werden, um damit anzuzeigen, dass nun eine Reihe von Optionen folgt.

Unter Linux gibt es darüber hinaus auch Optionen, die mit einem doppelten Minuszeichen beginnen und einen langen Optionsnamen aufweisen. Solche Optionen sind einerseits leichter lesbar als kurze, erfordern andererseits (gerade bei häufigen Aufrufen) auch mehr Schreibarbeit. Ein Beispiel für eine weit verbreitete lange Option ist `--version`; viele Programme geben bei einem Aufruf mit dieser Option die jeweilige Versionsnummer aus.

Eine Übersicht möglicher Optionen eines Befehls gibt die Manpage des jeweiligen Programms (`man programm`).

Argumente

Argumente dienen nicht zur Steuerung einer Anweisung, sondern liefern diesem Informationen, die es zu bearbeiten hat. Viele Anweisungen zur Manipulation von Dateien benötigen zum Beispiel die Namen der Dateien, die sie manipulieren sollen. Hier wird also nicht das Verhalten des Programmes geändert, sondern die Information variiert, die dem Programm für seine Arbeit zur Verfügung steht. Im Gegensatz zu Optionen kann es häufig eine sehr große Zahl verschiedener Argumente geben. Optionen hingegen sind immer nur in relativ beschränkter Zahl verfügbar – immer gerade so viele, wie der Programmierer in sein Programm implementiert hat.

Standard-Programme

Die folgenden Programme sind auf den meisten Debian-Systemen (und vielen anderen Linux-Distributionen) standardmäßig installiert. Die Liste beinhaltet eine Auswahl von Programmen, die einem mit großer Wahrscheinlichkeit früher oder später in Shell-Scripten begegnen und die ohne Superuser-Rechte genutzt werden können:

`alias`

Mit `alias kurzname='langer befehl'` lassen sich Abkürzungen für längere und/oder öfter gebrauchte Befehle definieren. Meist werden solche Aliase in der Konfigurationsdatei `~/.bashrc` definiert. Steht dort in einer Zeile beispielsweise `alias q='exit'`, so lässt sich das aktuelle Shellfenster wahlweise mit `q` oder mit `exit` schließen. Ist ein `alias ll='ls`

[1] Siehe *ls* für eine ausführlichere Beschreibung des Auflistungs-Programms.

`-lh'` definiert, so lässt sich eine ausführliche Liste des aktuellen Verzeichnisses mit `ll` aufrufen.

Persönlich verwende ich `alias`-Definitionen in sehr großem Umfang; beispielsweise habe ich vielerlei Abkürzungen zum schnellen Anwählen von Verzeichnissen definiert, beispielsweise `cdhg='cd ~/data/homepage/grund-wissen'`, `cdhgp='cd ~/data/homepage/grund-wissen/physik'` usw.

apt, aptitude

Mit `apt-cache search suchbegriff` beziehungsweise `aptitude search suchbegriff` kann das System nach Paketen durchsucht werden, deren Name oder Beschreibung dem Suchbegriff entspricht. Die Ausgabe von `aptitude` gibt zusätzlich an, ob die entsprechenden Pakete installiert sind (`i` für „installed“) oder nicht (`p` für „purged“).

Zum (De-)Installieren von Paketen sind Superuser-Rechte erforderlich; auf die Paketverwaltung mittels `apt` wird daher erst im Abschnitt *Administrations-Programme* ausführlich eingegangen.

basename

Mit `basename dateiname` wird der Dateiname (ohne den Verzeichnispfad) angezeigt. Mit `basename dateiname endung` wird zusätzlich zum Verzeichnispfad auch die angegebene Endung der Datei „abgeschnitten“ (nützlich für Konvertierungs-Skripte, automatische Umbenennungen, etc.).

Im umgekehrten Fall kann das Verzeichnis einer Datei mittels `dirname dateiname` bestimmt werden.

bc

Mit `bc` steht ein simpler Taschenrechner auf der Kommandozeile zur Verfügung. Zahlen und Rechenzeichen können nach dem Aufruf direkt eingegeben werden; bei Bedarf können auch Klammern gesetzt werden. Nach Bestätigung mit der `Enter`-Taste wird das Ergebnis berechnet und angezeigt. Frühere Berechnungen können mit der ↑-Taste wieder angezeigt beziehungsweise als Vorlage für eine neue Eingabe genutzt werden.

Ruft man `bc` mit der Option `-l` auf, so wird automatisch eine Standard-Bibliothek mit einigen wichtigen mathematischen Funktionen geladen. Beispielsweise kann so mittels `s(num)` der Sinus der Zahl `num` ausgegeben werden, mittels `c(num)` der Cosinus und mit `a(num)` der Arcus-Tangens; die übergebenen Werte `num` müssen dabei in Radiant angegeben werden. Mit `l(num)` kann der natürliche Logarithmus der Zahl `num` berechnet werden, mit `e(num)` wird der Wert der natürlichen Exponentialfunktion von `num` ausgegeben.

Man kann `bc` auch innerhalb von *Shell-Skripten* verwenden, um numerische Berechnungen auszuführen und die Ergebnisse gegebenenfalls in *Variablen* zu speichern. Dazu wird der auszuwertende Ausdruck mittels `echo` und einem *Pipe-Zeichen* an `bc` übergeben:

```
echo '8/3' | bc
# Ergebnis: 2.66666666666666666666

pi=$( echo 'a(1)*4' | bc -l )
echo $pi
# Ergebnis: 3.14159265358979323844
```

Beim letzten Beispiel als Trick genutzt, dass der Arcus-Tangens von 1 exakt den Wert $\frac{\pi}{4}$ ergibt; das Vierfache davon entspricht somit der Kreiszahl π. Dies kann auch genutzt werden, um (ohne Installation zusätzlicher Programme) einen einfachen CPU-Benchmark durchzuführen. Hierzu gibt man folgendes ein:

```
time echo "scale=5000; 4*a(1)" | bc -l
```

Über die `time`-Anweisung wird die Zeit ermittelt, die zum Ausführen eines Prozesses benötigt wird. Im konkreten Fall wird über `scale=5000` die Anzahl der zu ermittelnden Nachkomma-Stellen auf `5000` festgelegt und anschließend mittels `4*a(1)` der vierfache Wert der Arcus-Tangens-Funktion für den Wert `1` berechnet, was wiederum den Wert π liefert. Bei modernen Rechnern kann die Rechenzeit bei unter zwanzig Sekunden liegen, bei schwächeren Geräten kann die Rechenzeit auch rund hundert Sekunden betragen.

bzip2, bunzip2

Mit `bzip2 dateiname` kann eine Datei zu einer gleichnamigen Datei im `bz2`-Format im komprimiert werden. Mit `bunzip2 dateiname.bz2` kann die Datei wieder dekomprimiert werden.

Gibt man mehrere Dateinamen an, so wird jede Datei in ein eigenes Archiv komprimiert. Um eine einzelne `bz2`-komprimierte Datei zu erhalten, die mehrere Dateien enthält, so werden diese zunächst mittels `tar` zu einem Archiv gepackt.[1]

cat

Mit `cat file` wird der Inhalt einer Datei auf dem Bildschirm ausgegeben. Bei der Ausgabe von langen Dateien kann mit `Shift PageUp` und `Shift PageDown` auf- und abgeblättert werden. Der Anfang oder das Ende einer Datei kann optional auch mit `head file` beziehungsweise `tail file` angezeigt werden.

`cat` kann auch verwendet werden, um zwei Textdateien zu einer einzelnen zu verbinden („concatenate“). Um beispielsweise eine Textdatei an eine andere anzuhängen, lautet die Syntax `cat datei1 >> datei2`. Um aus zwei Dateien eine neue Datei zu erstellen, lautet die Syntax `cat datei1 datei2 > datei-neu.txt`.

[1] Mittels `tar -cjvf archivname datei1 datei2` können zwei oder mehrere Dateien direkt zu einem komprimierten `tar`-Archiv zusammgefasst werden.

cd

Mit `cd pfad` wechselt man zu einem bestimmten Verzeichnis. Die `pfad`-Angabe kann dabei absolut (ausgehende vom Basis-Verzeichnis `/`) oder relativ (ausgehend vom aktuellen Verzeichnis) sein.

- Mit `cd ..` gelangt man ins übergeordnete Verzeichnis, mit `cd ../..` in das nächst höhere usw.
- Mit `cd` gelangt man ebenso wie mit `cd ~` ins Home-Verzeichnis. Mit `cd ~benutzername` gelangt man (als Superuser) in das Home-Verzeichnis des angegebenen Benutzers.

Als Ergänzung zur `cd`-Anweisung können *alias*-Definitionen oder Programme wie *autojump* zum schnellen Wechseln von Verzeichnissen genutzt werden.

chmod

Mit `chmod` lassen sich die Zugriffsrechte einer Datei festlegen. Eine Datei kann lesbar (`r` für „read“), schreibbar (`w` für „write“) und/oder ausführbar (`x` für „executeable“) sein.

- Mit `chmod +x file` wird eine Datei (beispielsweise ein *Shell-Skript*) im aktuellen Ordner ausführbar gemacht. Sie kann anschließend mit `./file` aufgerufen werden.
- Mit `chmod -w file` werden der Datei die Schreibrechte entzogen.

...

clear

Mit `clear` wird der Bildschirm „geleert“ – lediglich der aktuelle Eingabe-Prompt bleibt bestehen. Die Shell-History (mit ↑ beziehungsweise ↓ abrufbar) bleibt unverändert.

cp

Mit `cp datei neuer-pfad` wird eine Datei (oder ein Verzeichnis) an eine andere Stelle kopiert. Es können mehrere Dateien auf einmal angeben werden; der zuletzt angegebene Pfad stellt dann den Zielpfad dar, in den alle zuvor angegebenen Dateien kopiert werden.

- Mit `cp -r` werden auch Unterverzeichnisse rekursiv kopiert (andernfalls werden sie weggelassen).
- Mit `cp -s` wird anstelle des Kopierens ein symbolischer Link am Zielpfad erstellt.

date

Mit `date` wird das aktuelle Datum und die aktuelle Uhrzeit angezeigt.

Die Ausgabe von `date` kann mittels verschiedener Aufrufparameter beliebig angepasst werden. Dies kann u.a. in Skripten hilfreich sein, um „Zeitstempel“ in Dateinamen aufzunehmen. Der Aufruf von `date +%Y%m%d_%H%M%S` gibt beispielsweise eine Zeitangabe im Format `YYYYMMDD_hhmmss` aus. Beispielsweise entspricht dem Datum „30.08.2012, 9:21:03 Uhr“ damit die Zeichenfolge „20120830_092103“.

df

Mit `df` wird angezeigt, wie viel Speicherplatz im Augenblick auf den eingehängten Laufwerken verfügbar ist („disk free“); `df` kann auch eine allgemeine Übersicht über eingehängte Laufwerke ausgeben

- Mit `df -h` wird die Ausgabe „human readable“ gestaltet, die Größen werden also in KB, MB oder GB anstelle von Bytes angegeben.
- Mit `df -aTh` erhält man eine ausführliche, aber gut lesbare Übersicht über alle eingehängten Dateisysteme und ihre Partitionen inklusive ihrer Device-Namen und Einhänge-Punkte (eine ähnliche Ausgabe erhält man mittels `mount | column -t`).

dirname

Mit `dirname dateiname` wird der Verzeichnisname einer Datei (ohne den eigentlichen Dateinamen) angezeigt.

du

Mit `du` wird angezeigt, wie viel Festplattenspeicher durch das aktuelle Verzeichnis und seiner Unterverzeichnisse belegt wird.

- Mit `du -h` wird die Ausgabe „human readable“ gestaltet, die Größen werden in also KB, MB oder GB anstelle von Bytes angegeben. * Mit `du -c` wird die Gesamtgröße jedes Unterverzeichnisses sowie des aktuellen Verzeichnisses ausgegeben. `du -S` bewirkt im Prinzip das gleiche, allerdings wird hierbei die Größe der Unterverzeichnisse nicht zur Berechnung einer Verzeichnisgröße einbezogen. * Mit `du -s` wird nur die Gesamtsumme der Dateigrößen ausgegeben. * Mit `du -L` wird statt der Größe von Symlinks die Größe der verlinkten Dateien berücksichtigt.

echo

Mit `echo variable` kann der Inhalt einer Variablen angezeigt werden. Beispielsweise liefert `echo $PATH` die Namen aller Verzeichnisse, in denen nach ausführbaren Shell-Programmen gesucht wird.

Möchte man `echo` verwenden, um einen Text mittels einer *Pipe* an ein anderes Programm zu übergeben, so muss beachtet werden, dass dabei beispielsweise das Newline-Zeichen `\n` nicht ausgewertet wird. Um dies zu erreichen, muss `echo` mit der Option `-e` aufgerufen werden.

exit

Mit `exit` wird der aktuelle Benutzer abgemeldet die aktuelle Sitzung (beispielsweise eine *SSH*-Verbindung) beendet. Ist nur eine Sitzung geöffnet, wird das Shell-Fenster geschlossen.

file

Mit `file dateiname` werden ausführliche Datei-Informationen (Dateityp, Version, Kodierung) der angegebenen Datei angezeigt.

find

Mit `find` können Verzeichnisse nach Dateien durchsucht werden. Angezeigt werden jeweils (nur) die Dateien, die dem vorgegebenen Suchmuster entsprechen. Die allgemeine Syntax lautet:

```
find basisordner kriterium [weitere kriterien]
```

Häufig genutzte Kriterien sind beispielsweise:

- `-name suchmuster`: Zeigt alle Dateien an, die dem Suchmuster entsprechen – einem „normalen“ Namen, oder einem regulären Ausdruck.
- `-iname suchmuster`: Zeigt alle Daten an, die dem Suchmuster entsprechen – Groß- und Kleinschreibung wird dabei ignoriert.
- `-mtime -n`: Zeigt alle Dateien an, die im Laufe der letzten n Tage (`n * 24h`) modifiziert wurden. Um Dateien anzuzeigen, deren letzte Änderung *mindestens* `n * 24h` zurückliegt, wird das `-`-Zeichen weggelassen.
- `-executeable`: Zeigt nur ausführbare Dateien an.
- `-size n [kMG]`: Zeigt nur Dateien an, deren Dateigröße über `n` Kilo-/Mega-/Giga-Bytes liegt.
- `-user name`: Zeigt nur Dateien an, die dem angegebenen Benutzer gehören.
- `-type [fdl]`: Zeigt Dateien an, die dem angegebenen Dateityp entsprechen (`f`: Normale Datei („file“), `d`: Verzeichnis („directory“), `l`: Symbolischer Link).

Mit `!-kriterium` können die obigen und weitere Kriterien (siehe `man find`) „umgekehrt“ werden, so dass sie die genau gegenteiligen Ergebnisse liefern.

`find` kann auch in Verbindung mit *grep* genutzt werden, um zunächst bestimmte Dateien zu finden, und diese dann nach bestimmten Inhalten zu durchsuchen. Um beispielsweise die Namen aller Python-Dateien (Endung `.py`) eines Verzeichnisses und aller Unterverzeichnisse auszugeben, welche die Zeichenkette `import sympy` beinhalten, kann man folgendes eingeben:

```
find ./ -name "*.py" -exec grep -l "import sympy" {} \;
```

Hierbei werden von `find`, ausgehend vom aktuellen Verzeichnis `./`, alle Dateien mit der Endung `.py` gesucht. Mit der Option `-exec` werden diese Dateien an das darauf folgende Programm übergeben, wobei die einzelnen Dateien an der Stelle eingefügt werden, wo die geschweiften Klammern `{}` stehen. Die `exec`-Anweisung muss am Ende mit `\;` abgeschlossen werden.

Noch einfacher ist die Verwendung von *xargs*, um die von `find` gefundenen Dateinamen an `grep` zu übergeben. Sollen beispielsweise alle Dateien mit der Endung `.rst` nach einem angegebenen Text durchsucht werden, kann folgendes eingegeben werden:

```
find ./ -name "*.rst" | xargs grep "Suchbegriff"
```

Nutzt man diese Kombination häufiger, so kann dafür in der Konfigurationsdatei `~/.bashrc` ein *alias* definiert werden:

```
alias rstgrep='find ./ -name "*.rst" | xargs grep'
```

Damit kann künftig `rstgrep` ebenso wie `grep` mit allen dort zur Verfügung stehenden Optionen aufgerufen werden.

free

Mit `free -h` bekommt man eine kurze Übersicht angezeigt, wieviel Arbeitsspeicher (RAM) aktuell belegt beziehungsweise noch frei ist.

grep

Mit `grep` („get regular expression“) können Eingabedaten, Textdateien oder Verzeichnisse nach beliebigen Suchbegriffen und regulären Ausdrücken durchsucht werden. Der allgemeine `grep`-Befehl hat folgende Syntax:

```
grep [optionen] suchmuster suchpfad
```

Möchte man beispielsweise das aktuelle Verzeichnis und alle Unterverzeichnisse rekursiv nach einem Suchbegriff durchsuchen, wobei die Groß- und Kleinschreibung ignoriert werden soll, so gibt man folgendes ein:

```
grep -lir "suchbegriff" ./
```

Mittels der Option `-l` werden nur die Dateinamen anstelle der zutreffenden Textzeilen ausgegeben, mit `-i` („ignore-case“) die Groß-und Kleinschreibung ignoriert, und mit `-r` („recursive“) die Durchsuchung der Unterverzeichnisse aktiviert. Mittels der Option `-c` wird die Anzahl an Treffern angezeigt, mit der Option `-v` werden diejenigen Zeilen als Ergebnis ausgegeben, auf die das Suchmuster *nicht* zutrifft.

Möchte man alle Ergebnisse anzeigen, die auf (mindestens) eines von mehreren angegebenen Suchmustern zutreffen, so können die einzelnen Suchmuster jeweils mit der Option `-e` angegeben werden.

Als *Exit-Status* liefert `grep` den Wert `0`, wenn die Suche erfolgreich war, `1`, wenn das Suchmuster nicht gefunden wurde, und `2`, wenn bei der Suche ein Fehler aufgetreten ist (beispielsweise eine Datei nicht lesbar war).

gzip, gunzip

Mit `gzip dateiname` kann eine Datei komprimiert, mit `gunzip dateiname.gz` wieder dekomprimiert werden. Mittels `zcat dateiname.gz` kann eine komprimierte Datei auf dem Bildschirm ausgegeben werden, ohne dass sie dazu auf der Festplatte entpackt wird (dies wird beispielsweise für Hilfeseiten genutzt, die in komprimierter Form auf der Festplatte abgelegt sind).

host

Mit `host URL` kann die IP-Adresse einer Webseite angezeigt werden. Beispiel:

```
host www.grund-wissen.de
# www.grund-wissen.de has address 188.40.57.88
```

Häufig wird `host` in Kombination mit dem Zusatz-Programm *whois* verwendet, um zusätzliche Informationen über den Server einer Domain zu erhalten.

hostname

Mit `hostname` kann der Netzwerk-Name des Rechners angezeigt werden, auf dem man aktuell eingelogt ist. Dies kann beispielsweise nützlich sein, wenn man via *ssh* auch auf externen Rechnern arbeitet.

inxi

Mit `inxi` kann man sich Informationen über das aktuell installierte Betriebsystem sowie einige Hardware-Informationen anzeigen lassen. Ruft man `inxi -Fx` auf, so erhält man eine detailierte, aber gut lesbare Liste.

Die wohl detailierteste Liste mit Hardware-Informationen liefert wohl `dmidecode`; da die Ausgabe recht lang ist, empfiehlt es sich, diese mittels `dmidecode | less` im Pager *less* zu betrachten.

ip

Mit `ip r` kann die lokale Netzwerkadresse (`192.168.xxx.xxx`) angezeigt werden.

kill, killall

Mit `kill prozessID` beziehungsweise mit `killall programmname` lässt sich ein (evtl. außer Kontrolle geratenes) Programm beenden. Die ID eines Prozesses lässt sich beispielsweise mit `ps -aux programm`, `pgrep programmname` oder mittels des Systemmonitors *top* anzeigen. Innerhalb von `top` lässt sich der `kill`-Befehl mittels `k` starten.

- Mit `kill -9 prozessID` beziehungsweise mit `killall -9 programmname` wird ein Prozess unterbrochen, egal welchen Signalwert er gerade ausgibt. Dies ist die stärkste Form, einen unerwünschten Prozess zum erliegen zu bringen.

less

Mit `less dateiname` kann der Inhalt einer Textdatei angezeigt werden. Die Anzeige beginnt am Anfang der Datei, mit der ↓ beziehungsweise ↑ kann innerhalb der Datei nach unten beziehungsweise oben gescrollt werden. Mit `/` kann die Datei nach einem Begriff durchsucht werden, mit `n` kann man zum nächsten Suchergebnis springen. Mit `q` wird less wieder beendet.

ln

Mittels `ln` beziehungsweise `ln -s` lassen sich die zwei unter Linux möglichen Arten von Verknüpfungen erzeugen:

- Mit `ln datei1 datei2` wird zu einer existierenden Datei `datei1` die Datei `datai2` als so genannter „Hardlink“ erzeugt. Dabei handelt es sich im Grunde um eine zusätzliche Bezeichnung für die selbe Speicherstelle auf der Festplatte. Um beispielsweise eine mit Hardlinks versehene Datei zu löschen, müssen ebenfalls sämtliche Hardlink entfernt werden, um die Speicherstelle freizugeben.

  Da sich Hardlinks stets auf der gleichen Partition befinden müssen wie die Original-Dateien und sich nur auf „normale“ Dateien, jedoch nicht auf Verzeichnisse anwenden lassen, werden sie unter Linux nur selten verwendet.

- Mit `ln -s datei1 datei2` wird zu einer existierenden Datei `datei1` die Datei `datai2` als so genannter „Symbolischer Link“ erzeugt (auch „Symlink“ oder „Softlink“ genannt). Dabei handelt es sich um eine neue Datei, deren einziger Inhalt ein Verweis auf die bestehende Datei ist (Symlinks sind daher stets nur wenige Bytes groß).

  Wird ein Symlink zu einer Textdatei mit einem Editor geöffnet, verändert und gespeichert, so wird auch die Originaldatei entsprechend verändert. Wird allerdings der Symlink einer Datei gelöscht, so bleibt die Originaldatei bestehen. Wird im umgekehrten Fall die Original-Datei gelöscht oder umbenannt, so bleibt der Symlink als Datei bestehen, zeigt aber ins Leere („gebrochener Link“). Der Symlink muss in diesem Fall entfernt und neu erzeugt werden.[2]

[2] Beispielsweise bietet der Dateimanager *Midnight Commander* die Tastenkombination `Ctrl x Ctrl s` zur schnellen Erzeugung von Symlinks an. Mit dem Midnight Commander oder mittels `cp -L` kann

Wird ein Symlink zu einer ausführbaren Datei erzeugt, so kann diese auch über den Symlink aufgerufen werden. Wird ein Symlink zu einem Ordner erstellt, so lassen sich dessen Inhalte auch über den Symlink anzeigen und verändern. Da Symlinks auch anders benannt sein können als die Originaldateien, können sie beispielsweise dazu genutzt werden, um aus einer vorhandenen Musiksammlung individuelle Playlisten in separaten Ordnern anzulegen.

Mit Symlinks verknüpfte Ordner beziehungsweise Dateien müssen nicht zwingend auf dem gleichen Datenträger beziehungsweise der gleichen Partition liegen. So ist es beispielsweise möglich auf einen (automatisch ins `/media`-Verzeichnis eingebundene) USB-Stick oder eine verschlüsselte Festplattenpartition über einen entsprechenden Symlink im Home-Verzeichnis zuzugreifen.

`locate`

Mit `locate suchbegriff` werden alle Dateinamen des Systems nach einem Suchbegriff durchsucht und die Ergebnisse angezeigt. Mit `locate -i suchbegriff` wird dabei die Groß- und Kleinschreibung ignoriert.

Um auch neueste Änderungen, die sich seit dem letzten Systemstart ergeben haben, anzuzeigen, kann die Datei-Datenbank mittels `updatedb` aktualisiert werden.

`ls`

Mit `ls` wird der Inhalt des aktuellen Verzeichnisses ausgegeben. Mit weiteren Parametern lässt sich die Ausgabe den eigenen Wünschen anpassen:

- `ls -a` zeigt auch Konfigurationsdateien und -verzeichnisse an, also Dateien, deren Name mit einem `.` beginnt („list all“).
- `ls -lh` liefert eine ausführliche Liste, die auch Informationen über Dateityp, Dateirechte, Modifikationszeit und Dateigröße beinhaltet („long list“). Der Zusatz `h` bewirkt, dass die Dateigröße „human readable“, also in KB, MB oder GB anstelle von Bytes angeben wird.
- `ls -r` listet rekursiv die Inhalte des aktuellen Verzeichnisses und der darin enthaltenen Unterverzeichnisse auf.

Mit `ls pfad` kann ebenfalls der Inhalt eines anderen Verzeichnisses ausgegeben werden, ohne dass das aktuelle Verzeichnis verlassen wird.

Eine vollständige Beschreibung aller Optionen findet sich in den Manpages (`man ls`).

---

man darüber hinaus beim Kopieren von Symlinks optional wieder auf die Originaldateien zurückgreifen und deren Inhalte kopieren.

Wird ein Symlink kopiert, so zeigt auch die Kopie auf den gleichen (absoluten) Pfad wie der ursprüngliche Symlink.

`lsblk`, `lsusb` und `lscpu`

Mit `lsblk` werden alle „Block-Devices“, also Datenträger aufgelistet, die vom System erkannt werden. Zu jedem Device (beispielsweise `sda`, `sdb` usw.) werden zudem die einzelnen Partitionsnamen, die Größe der Partitionen sowie gegebenenfalls die Einhänge-Punkte („Mountpoints“) ausgegeben.

Mit `lsusb` werden Informationen über alle verfügbaren USB-Ports beziehungsweise dort angeschlossene Geräte ausgegeben.

Mit `lscpu` werden Informationen über die CPU ausgegeben; unter anderem kann man daran ablesen, ob es sich beim aktuellen Rechner um einen 32bit- oder 64bit-System handelt.

Das ähnliche Programm *lshw*, das allgemeine Hardware-Informationen anzeigt, sollte nur mit Root-Rechten aufgerufen werden.

`man`

Mit `man programm` werden die Hilfeseiten („Manual-Pages“, „Manpages“) eines Programms angezeigt. Hier werden sämtliche Programm-Aufruf-Optionen sowie meist einige nützliche Beispielfälle beschrieben.

Beispiel: Mit `man less` werden die Hilfe-Seiten zum Pager-Programm „less“ angezeigt.

Um alle Hilfeseiten nach einem bestimmten Begriff zu durchsuchen, kann `man` mit der Option `-k` („keyword“) aufgerufen werden:

Beispiel: Mit `man -k find` werden alle Programmnamen und Funktionen aufgelistet, die den Suchbegriff „find“ in ihrer Hilfeseite enthalten.

`mkdir`

Mit `mkdir verzeichnisname` wird ein neues Verzeichnis angelegt. `verzeichnisname` kann auch ein absoluter Pfad sein, dann wird das Verzeichnis an entsprechender Stelle angelegt.

`mv`

Mit `mv datei neuer-pfad` wird eine Datei (oder ein Verzeichnis) an eine andere Stelle verschoben. Es können mehrere Dateien auf einmal angeben werden; der zuletzt angegebene Pfad stellt dann den Zielpfad dar, in den alle zuvor angegebenen Dateien verschoben werden.

Mit `mv alter-dateiname neuer-dateiname` lässt sich eine Datei umbenennen.

passwd

Mit `passwd` kann man das zum eigenen Account gehörende Passwort ändern. Man wird nach einem neuen Passwort gefragt, wobei dieses sicherheitshalber zwei mal eingegeben werden muss.

pdftotext

Mit `pdftotext pdf-datei` lässt sich der gesamte Text einer PDF-Datei in eine Textdatei extrahieren. Der Text lässt sich dann meist recht einfach mittels ein paar Vim-Tricks und Einfügen von Restructured-Text-Syntax via Sphinx in ein druckbares Latex-Dokument oder eine leicht durchsuchbare HTML-Seite umwandeln. Sehr nützlich!

pwd

Mit `pwd` („print working directory“) wird der volle Pfad des aktuellen Verzeichnisses ausgegeben.

rm

Mit `rm datei(en)` lässt sich eine oder mehrere Datei(en) unwiderruflich löschen.

- Mit `rm -r verzeichnis/*` werden rekursiv alle Inhalte, ausgehend von `verzeichnis` gelöscht.

Achtung: Die Shell kennt keinen „Papierkorb“, Löschvorgänge sind somit endgültig. Vor dem Löschen sollte man sich daher stets vergewissern, ob man die entsprechenden Dateien auch wirklich löschen möchte.

Mit regulären Suchmustern wie `*` ist beim Löschen stets besondere Vorsicht geboten: Während `rm -r *~` ausgehend vom aktuellen Verzeichnis alle (von manchen Editoren angelegten) temporären Dateien löscht, würde `rm -r ~*` sämtliche Inhalte des Home-Verzeichnisses unwiderruflich löschen!

rmdir

Mit `rmdir verzeichnisname` wird ein Verzeichnis gelöscht, sofern es leer ist. Möchte man ein nicht-leeres Verzeichnis löschen, so empfiehlt sich das einfach zu tippende `rm -r verzeichnisname*` (es werden rekursiv alle Dateien, deren Pfadname mit `verzeichnisname` beginnt, gelöscht).

rsync

Mit `rsync quelldatei backupdatei` kann man eine Datei oder ein Verzeichnis gegenüber einer Backup-Kopie der Datei beziehungsweise des Verzeichnisses aktualisiert („syn-

chronisiert“) halten. Der Backup findet dabei nur in eine Richtung statt, `rsync` prüft also anhand der letzten Bearbeitungszeit (`MTIME`) einer Datei, ob sich in der Quelle gegenüber dem Backup eine Veränderung ergeben hat. Falls ja, werden diese Änderungen übernommen.

Möchte man einen Backup von einem ganzen Verzeichnispfad mitsamt allen Unterverzeichnissen anlegen oder aktuell halten, so empfiehlt sich folgender Aufruf von `rsync`:

```
rsync -vahz quellverzeichnis/* zielverzeichnis
```

Das Zielverzeichnis muss dabei ein bestehender Ordner sein, kann sich allerdings auch auf einer anderen Partition, einem externen Datenträger, oder – bei Verwendung von `ssh` – sogar auf einem anderen Rechner befinden.

ssh

Mit `ssh benutzername@rechneradresse` kann man sich auf einem anderen Linux-Rechner im lokalen Netzwerk oder im Internet anmelden. Ist die Rechneradresse erreichbar, erscheint ein Dialogfeld zur Passworteingabe. Alle auf dem Fremdrechner verfügbaren Shell-Befehle und -Programme lassen sich somit „ferngesteuert“ ausführen.

Zu einer Verwendung von `ssh` im lokalen Netzwerk sollten die folgenden beiden Pakete installiert werden:

```
sudo aptitude install openssh-client openssh-server
```

Die Benutzung von `ssh` ist im Abschnitt *SSH – Arbeiten auf entfernten Rechnern* ausführlich beschrieben.

tar

Mit `tar` können mehrere Dateien zu einem Archiv zusammengefasst werden. Beispiel:

```
tar -cf archiv.tar datei1 datei2 ...
```

Mit der Option `z` wird das Archiv zusätzlich mit `gzip` komprimiert. Mit der Option `v` wird der Fortschritt der Archivierung angezeigt („verbose“ = redselig).

```
tar -czvf archiv.tar.gz datei1 datei2 ...
```

Der Inhalt eines `tar`-Archivs kann mittels `tar tf archiv.tar` angezeigt werden. Mittels der Option `x` (extract) kann der Inhalt des Archivs wieder entpackt werden:

```
tar -xvf archiv.tar          # für "normale" Archive
tar -xvzf archiv.tar.gz      # für komprimierte Archive
```

Anstelle der Option `-z` kann auch `-j` eingegeben werden, um anstelle der `gz`-Komprimierung das stärker komprimierende `bz2`-Format zu nutzen.

tee

Das Programm `tee` liest Text von der Standardeingabe ein und schreibt diesen sowohl auf die Standardausgabe als auch in eine Datei. Dies kann beispielsweise genutzt werden, um eine Fehlermeldung einerseits auf dem Bildschirm auszugeben, andererseits gleichzeitig aber auch einen Eintrag in einer Logdatei zu erstellen. Meistens wird dazu ein Text mittels `echo` ausgegeben und mittels des *Pipe-Operators* `|` an `tee` weitergereicht, beispielsweise `echo "Hallo!" | tee dateiname`.

top

Mit `top` werden aktuell laufende Prozesse, geordnet nach CPU-Auslastung, angezeigt. Auf diese Weise kann die Prozess-ID (PID) eines außer Kontrolle geratenen Programms oder „Speicherfressers“ schnell ausfindig gemacht und der entsprechende Prozess abgebrochen werden.

- Mit `P` werden die Prozesse nach CPU-Auslastung sortiert, mit `M` nach Memory-Auslastung, mit `N` nach Prozess-ID.
- Mit `k` wird nach Eingabe einer PID-Nummer der entsprechende Prozess abgebrochen. Die Nachfrage, mit welchem Signal der Prozess unterbrochen werden soll (Vorgabewert: `15`), kann meist mit `Enter` bestätigt werden. Bei hartnäckigen Prozessen kann `9` angegeben werden, um den Prozess unabhängig von dessen Signalwert abzubrechen.
- Mit `q` wird `top` wieder beendet.

touch

Mit `touch dateiname` lässt sich eine neue, leere Datei (beispielsweise eine neue Log-Datei) anlegen.

uname

Mit `uname -a` kann angezeigt werden, ob es sich bei dem laufenden System um ein 32- oder ein 64-bit System handelt. Dies muss bisweilen berücksichtigt werden, wenn Zusatzpakete nicht via *aptitude* installiert werden, sondern beispielsweise als fertige Debian-Pakete von einer Webseite heruntergeladen werden sollen.

Mit `uname -mrs` bekommt man die aktuelle Version des Kernels angezeigt.

wc

Mit `wc dateiname` wird die Anzahl der Zeilen, Worte und Zeichen ausgegeben, die in der angegebenen Datei vorkommen („word count“).

Oftmals wird `wc` in Kombination mit `find` oder `grep` verwendet, um die Anzahl von Treffern bei einer bestimmten Suche anzuzeigen; um beispielsweise die Anzahl aller regulären Dateien des aktuellen Verzeichnisses mitsamt aller Unterverzeichnisse (ohne die Verzeichnisnamen selbst) anzuzeigen, kann man `find ./ -type f | wc` eingeben.

wget

Mit `wget` lassen sich mit wenig Aufwand Downloads von der Shell aus starten. Dabei können ganze Verzeichnisse (falls gewünscht auch mitsamt Unterverzeichnissen), bestimmte Dateitypen, Dateigrößen usw. als Auswahlkriterien festgelegt werden.

Um beispielsweise alle Beispiel-Dateien (1000 Stück!) des Computer-Algebra-Systems Maxima von der Seite http://www.lungau-academy.at/wx1/ herunterzuladen, genügt folgender Befehl:

```
wget -r -l1 -np -A wxmx http://www.lungau-academy.at/wxmax1001/
```

Hierbei steht `-r` für ein rekursives Herunterladen, `-l1` beschränkt die Anzahl der durchsuchten Unterverzeichnisse auf 1. Die Option `-np` beziehungsweise `--no-parent` ist wichtig, um zu verhindern, dass auch übergeordnete Verzeichnisse durchsucht werden – dies könnte im Zweifelsfall die Downloadmenge erheblich vergrößern. Die Option `-A filetype` legt anhand der angegebenen Dateiendung(en) fest, welche Datentypen akzeptiert werden (im umgekehrten Fall können mit `-R filetype` bestimmte Datentypen zurückgewiesen werden).

Ein gutes Tutorial (en.) findet sich hier.

which

Mit `which programm` wird angezeigt, unter welchem Systempfad die auszuführende Datei des angegebenen Programms zu finden ist.

xargs

Mit `xargs` können die Ergebnisse eines Shell-Programms als Argumente eines anderen Shell-Programms verwendet werden. Dies ist beispielsweise bei der Kombination von *find* und *grep* nützlich, um die von `find` gefundenen Dateinamen nicht unmittelbar als (Eingabe-)Text, sondern als Zieldateien nach bestimmten Mustern zu durchsuchen.

Sollen beispielsweise alle `.tex`-Dateien nach einem bestimmten Begriff durchsucht werden, kann man folgendes eingeben:

```
find ./ -name "*.tex" | xargs grep
```

Ohne die Verwendung von `xargs` würden hier nur die Namen der Dateien, jedoch nicht deren Inhalt durchsucht.

zip, unzip

Mit `zip` können mehrere Dateien zu einem Datei-Archiv gebündelt, mit `unzip` wieder entpackt werden. Die grundlegenden Befehle sehen etwa so aus (weitere Informationen erhält man mittels `man zip` beziehungsweise `man unzip`):

- Mit `zip archivname.zip datei1 datei2 ...` werden mehrere Dateien zu einem (komprimierten) `zip`-Archiv gebündelt. Mit `zip -r` können Dateien und/oder Verzeichnisse rekursiv (samt Unterverzeichnissen) gepackt, mit `zip -g` zu einem bestehenden Archiv hinzugefügt werden.
- Mit `unzip archivname` wird ein `zip`-Archiv wieder entpackt.

Zusätzliche Shell-Programme

Die folgenden, meiner Meinung nach durchaus nützlichen Programme sind auf frisch installierten Systemen zwar (meist) nicht von Grund auf enthalten, lassen sich jedoch einfach mittels der Paketverwaltung nachinstallieren.

abcde

abcde steht für „A better CD encoder“ und bietet in der Tat eine elegante Möglichkeit, eine Audio-CD automatisch als `ogg` oder `mp3`-Dateien einzulesen.

`abcde` ist über die Paketverwaltung mittels des gleichnamigen Pakets installierbar:

```
sudo aptitude install abcde
```

Nach dem Einlegen einer Audio-CD und dem Aufruf von `abcde` wird eine CD-Datenbank durchsucht und gegebenenfalls passende Einträge angezeigt. Nach Wunsch können die Meta-Daten noch bearbeitet werden, ansonsten wird nach Bestätigung mit `Enter` automatisch der Einlese- und Kodierungsprozess gestartet.

antiword

Mit `antiword` lassen sich MS-Word-Dokumente (`.doc`) bequem in einem Shell-Fenster als Textform anzeigen.

`antiword` kann mittels der Paketverwaltung `aptitude` einfach installiert werden:

```
sudo aptitude install antiword
```

Die Ausgabe eines Word-Dokuments auf dem Bildschirm lässt sich dann folgendermaßen erreichen:

```
antiword file.doc
```

Die Ausgabe kann selbstverständlich auch in eine Textdatei umgeleitet werden:

```
antiword file.doc > file.txt
```

Das erspart oftmals ein Öffnen von Libre-Office oder *Abiword*. Leider kann `antiword` keine neueren `docx`-Dateien als Text ausgeben; hierzu kann allerdings das Paket `docx2txt` mittels *apt* installiert werden, das ein gleichnamiges Programm zum Abbilden einer `docx`-Datei in eine gleichnamige `.txt`-Datei genutzt werden kann.

autojump

Mit `autojump` lassen sich häufig besuchte Verzeichnisse in einer Shell schnell und einfach ansteuern.

`autojump` ist über die Paketverwaltung mittels des gleichnamigen Pakets installierbar:

```
sudo aptitude install autojump
```

Nach der Installation muss noch folgender Eintrag in die Konfigurationsdatei `~/.bashrc` beziehungsweise `~/.zshrc` eingefügt werden:

```
. /usr/share/autojump/autojump.sh
```

Öffnet man dann eine neue Shell, so werden alle besuchten Verzeichnisse automatisch und auch über die aktuelle Shell-Sitzung hinaus in einer „History“ gespeichert; je häufiger ein Verzeichnis besucht wurde, desto höher wird sein Ranking in dieser History. Um in ein Verzeichnis zu wechseln, das einen Eintrag in dieser History hat, genügt es `autojump` mit einem Teil des Pfadnamens aufzurufen; für `autojump` wird bei der Installation zugleich automatisch das Alias `j` definiert.

Beispiel:

```
# In ein "neues" Verzeichnis wechseln:
cd /etc/init.d

# Zurück ins Home-Verzeichnis wechseln:
cd

# Mit autojump wieder in das obige Verzeichnis wechseln:
j init
```

Durch einen Aufruf von `j --stat` kann man sich anzeigen lassen, welche Häufigkeitswertung die einzelnen bisher besuchten Verzeichnisse haben. Ich persönlich habe mir für diesen Aufruf, da ich ihn für ziemlich nützlich halte, ebenfalls ein Alias in der Konfigurationsdatei `~/.zshrc` definiert:

```
alias J='autojump --stat'
```

Damit kann die Verzeichnis-History in Kombination mit *grep* einfach durchsucht werden:

```
J | grep stichwort

# Beispiel:
J | grep -i init

# Ergebnis:
14.1:   /etc/init.d
```

Was `autojump` erst Recht zu einem sehr nützlichen Alltags-Werkzeug macht, ist dass auch mehrere Pfad-Teile angegeben werden können:

```
# In ein "neues" Verzeichnis wechseln:
cd /usr/local/lib

# Zurück ins Home-Verzeichnis wechseln:
cd

# Mit autojump wieder in das obige Verzeichnis wechseln:
j usr lib
```

Autojump gleicht also einem Fuzzy-Finder für Pfadangaben: Einmal besucht, können Verzeichnisse auf diese Weise (sogar ohne Berücksichtigung von Groß- und Kleinschreibung) schnell wiedergefunden und angewählt werden. Autojump kann somit als Ergänzung zur *cd*-Anweisung einige Tipp-Arbeit sparen.

cmus

Der Console-Music-Player `cmus` bietet eine schlichte und übersichtliche Bedienoberfläche, um alle gängigen Audio-Formate (`ogg`, `mp3`, `wav`, `flac`, `aac`) sowie Playlisten (`m3u` und `pls`) innerhalb eines Shell-Fensters abzuspielen. Der Player bietet nicht so viele Möglichkeiten wie beispielsweise *audacious*, benötigt dafür aber keine graphische Oberfläche.

`cmus` ist über die Paketverwaltung mittels des gleichnamigen Pakets installierbar:

```
sudo aptitude install cmus
```

Anschließend kann der Player durch den Aufruf von `cmus` gestartet werden. Mittels der Tasten `1` bis `7` kann zwischen verschiedenen Ansichten gewechselt werden:

1	Bibliothek	Zweispaltige Ansicht: Links werden Künstler und Album aufgelistet, rechts die jeweiligen Lieder
2	Sortierte Bibliothek	(Flache Listenansicht aller Lieder, mit der Möglichkeit, die Sortierreihenfolge selbst festzulegen)
3	Wiedergabeliste	Anzeige der (editier- und speicherbaren) Wiedergabeliste
4	Warteliste (Queue)	Anzeige der unmittelbar abzuspielenden Lieder
5	Datei-Browser	Dateisystemansicht mit der Möglichkeit zum Hinzufügen von Liedern zur Sammlung, der Wiedergabeliste oder Warteliste
6	Datei-Filter	Anzeige benutzerdefinierter Filter
7	Einstellungen	Mit `d` kann man Einstellungen löschen, mit `Enter` modifizieren sowie mit `Leertaste` konkrete Variablen ändern.

Beim erstmaligen Starten von `cmus` sollte zunächst ein Verzeichnis mit Audio-Dateien in die Bibliothek geladen werden. Hierfür wechselt man mittels : auf die Kommandozeile und gibt dort folgende Anweisung ein:[1]

```
:add ~/Musik
```

Nachdem die Sammlung eingelesen wurde, werden die Lieder in den Bibliotheks-Ansichten `1` und `2` den Namen der Interpreten sortiert angezeigt.

In den einzelnen Ansichten (`1` bis `5`) können mittels der jeweiligen Tasten folgende Funktionen aufgerufen werden:

[1] `cmus` speichert alle Angaben zu den eingelesenen Dateien in der Datei `~/.cmus/cache`. Um das Programm schlank und schnell zu halten, wird diese Datei nicht kontinuierlich aktualisiert. Dies hat zur Folge, dass `cmus` nicht erkennt, wenn die Metadaten eines Stücks von einem anderen Programm (wie beispielsweise `EasyTAG`) verändert wurden. Damit die Änderungen in `cmus` angezeigt werden, muss zunächst die Bibliothek mit `:clear -l` gelöscht und die Cache-Datei mit `u` aktualisiert werden. Anschließend kann man die Bibliothek neu einlesen.

`Enter`	Datei abspielen beziehungsweise Verzeichnis öffnen
`c`	Pause-Modus an- und ausschalten („continue“)
`b`	Nächsten Titel abspielen
`/`	nach Suchmuster in Dateinamen oder ID-Tags suchen
`n`	zur nächsten Datei gehen, auf die Suchmuster zutrifft
`N`	zur vorherigen Datei gehen, auf die Suchmuster zutrifft
`y`	Datei oder Verzeichnis unter Cursor zur Wiedergabeliste (`3`) hinzufügen
`e`	Datei oder Verzeichnis unter Cursor an die Warteliste (`4`) anfügen
`E`	Datei oder Verzeichnis unter Cursor an den Anfang der Warteliste (`4`) setzen
`a`	Datei oder Verzeichnis unter Cursor in die Bibliothek (`1` beziehungsweise `2`) aufnehmen
`-`	Lautstärke um 10% herabsetzen
`+`	Lautstärke um 10% erhöhen
`,`	Aktuell abgespielte Datei 1 Minute zurückspulen
`.`	Aktuell abgespielte Datei 1 Minute vorspulen
←	Aktuell abgespielte Datei 5 Sekunden zurückspulen
→	Aktuell abgespielte Datei 5 Sekunden vorspulen

In der Infozeile (vorletzte Zeile auf dem Bildschirm) werden auf der rechten Seite Infos über die aktuellen Wiedergabeoptionen (Zufallswiedergabe, Wiederholung usw.) eingeblendet. Diese können folgendermaßen verändert werden:[2]

`s`	Zufallswiedergabe aktivieren oder deaktivieren
`r`	Wiedergabe-Modus (der ganzen Playliste beziehungsweise des aktuellen Albums) aktivieren oder deaktivieren
`Ctrl r`	Wiederholung des aktuellen Lieds aktivieren oder deaktivieren

Um Dateien innerhalb der Wiedergabeliste oder Warteliste nach oben oder unten zu verschieben, können die Tasten `p` und `P` („push“) genutzt werden:

`p`	Datei unter Cursor in der Ansicht `3` oder `4` nach unten verschieben
`P`	Datei unter Cursor in der Ansicht `3` oder `4` nach oben verschieben

Markiert man hierbei zunächst mehrere Dateien mittels der `Space`-Taste, so können diese anschließend mittels `p` oder `P` hinter beziehungsweise vor die Datei unter dem Cursor verschoben werden. Mit `D` oder `Del` können Dateien wieder aus der Wiedergabe- oder Warteliste entfernt werden.

Mehr Infos gibt es unter CMUS im Ubuntuuser-Wiki.

[2] Zudem kann mit `M` festgelegt werden, ob nach dem Abspielen aller Titel der Warteliste weitere Titel der Bibliothek wiedergegeben werden sollen. Persönlich habe ich diese Option grundsätzlich abgeschaltet, so dass in der Infozeile rechts nicht „album from library“, sondern „playlist“ steht.

fdupes

Mit `fdupes` kann man ein Verzeichnis nach doppelten Dateien durchsuchen und diese gegebenenfalls auflisten.

Das Programm kann mittels `apt` über das gleichnamige Paket installiert werden:

```
sudo aptitude install fdupes
```

Aufgerufen wird `fdupes` mit folgender Syntax:

```
fdupes verzeichnis
```

Verglichen werden die Dateien des Verzeichnisses dabei zunächst anhand ihrer Größe, anschließend anhand eines Byte-für-Byte-Vergleichs; `fdupes` erkennt damit auch identische, aber unterschiedlich benannte Dateien (unabhängig vom Dateiformat). Mit `fdupes -r verzeichnis` werden auch die Unterverzeichnisse rekursiv mit durchsucht, mit der Option `-s` können zusätzlich auch Symlinks zu Verzeichnissen berücksichtigt werden.

feh

Feh ist ein kleines, aber feines Bildbetrachtungs-Programm. Es kann über das gleichnamige Paket mittels der Paketverwaltung installiert werden:

```
sudo aptitude install feh
```

`feh` setzt eine aktive graphische Oberfläche voraus, kann also nicht als „reines“ Shell-Programm (ohne X-Server) verwendet werden. (Für derartige Anwendungen kann jedoch auf das Framebuffer-Image-Programm `fbi` zurückgegriffen werden.)

Der grundlegende Aufruf von `feh` sieht folgendermaßen aus:

```
feh image-file
```

Um alle Bilder des aktuellen Verzeichnisses anzusehen, genügt folgender Aufruf:

```
feh *
```

`feh` kann auf einfache Weise über die Tastatur gesteuert werden:

- Mittels der Pfeiltasten ← und → kann, sofern mehrere Bilder geöffnet werden, zum vorherigen beziehungsweise nächsten Bild gewechselt werden.
- Mittels der Pfeiltasten ↑ und ↓ wird in das Bild hinein- beziehungsweise herausgezoomt.
- Mittels `v` kann zwischen einer Vollbild- und einer normalen Anzeige gewechselt werden.
- Mittels < und > kann ein Bild gegen beziehungsweise mit dem Uhrzeigersinn um 90° gedreht werden.

- Mittels `q` („quit“) wird `feh` beendet.

`feh` kann mit einer Vielzahl an Parametern aufgerufen werden, um beispielsweise die geöffneten Bilder als Slideshow wiederzugeben. Hilfreich sind insbesondere die beiden Parameter `-d` und `-F`, mittels derer `feh` automatisch im Vollbild-Modus startet (`feh -F`) und den aktuellen Datei-Namen anzeigt (`feh -d`). Um beide Optionen als Standard zu definieren, bietet sich in der Konfigurationsdatei `~/.bashrc` folgendes Alias an:

```
alias feh='feh -d -F'
```

Nach einem neuen Laden der Konfigurationsdatei (`source ~/.bashrc`) beziehungsweise in jedem neu geöffneten Shell-Fenster wird `feh` anschließend automatisch mit den beiden obigen Parametern gestartet.

Weitere Infos finden sich in den `man`-Pages und unter Feh im Ubuntu-Wiki.

gpg

GPG (GNU Privacy Guard) ist ein freies Verschlüsselungssystem, welches hauptsächlich für die Verschlüsselung von Emails benutzt wird. GPG gehört zu allen Standard-Linux-Distributionen und ist auch auf einer Vielzahl anderer Systeme lauffähig.

Das Programm `gpg` ist im Abschnitt *gpg – Signieren und Verschlüsseln von Dateien* ausführlich beschrieben.

imagemagick

Bei Imagemagick handelt es sich um eine Sammlung von mehreren kleinen Bildbearbeitungs-Programmen, mit deren Hilfe einfache Anpassungen von Graphiken – beispielsweise Formatumwandlungen, Erzeugung von kleinen Vorschaubildern, Fotomontagen u.ä. – stapelweise als Shell-Skript auf eine Vielzahl von Dateien anwenden lassen.

Sollte `imagemagick` nicht bereits installiert sein, lässt es sich einfach mittels `aptitute` nachinstallieren:

```
sudo aptitute install imagemagick
```

Die Imagemagick-Suite umfasst folgende Bildbearbeitungs-Programme:

- `import`
- `convert`
- `montage`
- `display`

Diese kleinen Hilfsprogramme sind nützlich, um automatisiert bestimmte Bildbearbeitungen als Skript auszuführen.

Beispiele:

- Umwandeln eines transparenten in einen weißen Hintergrund:

```
convert image-old.png -background white -flatten -alpha off image-new.png
```

- Zusammenfügen mehrere Bilder vertikal zu einem „Filmstreifen“:

```
montage -mode concatenate -tile 1x in-*.jpg out.jpg
```

- Verkleinern von Digitalkamera-Fotos auf kleine Formate, beispielsweise Fotos von Rezepten für's Vegan-Kochbuch:

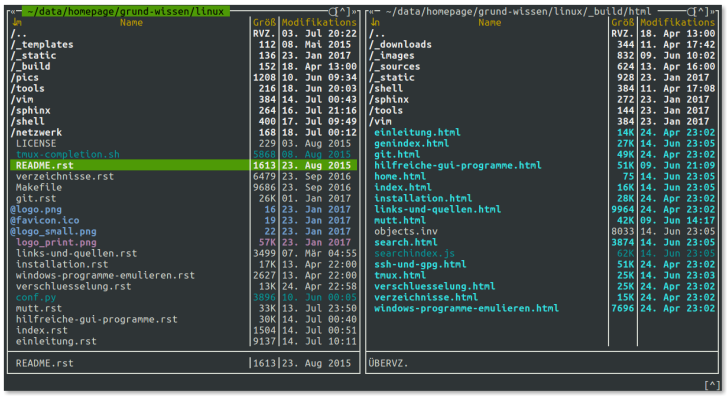
```
for i in *.jpg ; \
    do convert $i -resize '1200x800 -quality 80 $(basename $i .jpg).png ; \
    done
```

  Das obige Mini-Skript wandelt alle `.jpg`-Dateien des aktuellen Verzeichnisses in `1200 px` breite PNG-Dateien gleichen Namens um. Die Dateien können anschließend mit *pngnq* weiter komprimiert werden.

Mehr Infos zu `imagemagick` gibt es unter Imagemagick im Ubuntu-Wiki.

mc

Der „Midnight Commander“ `mc` gilt als einer der praktischsten Dateimanager überhaupt. Er bietet eine klar strukturierte Bedienoberfläche und ermöglicht es, Dateien mit nur wenigen Tastendrücken schnell und elegant zu verwalten.



`mc` ist über die Paketverwaltung mittels des gleichnamigen Pakets installierbar:

```
sudo aptitude install mc
```

In der Grundeinstellung sind im Midnight-Commander zwei Ordner-Panele nebeneinander angeordnet. Zwischen den beiden Panelen (und damit zwischen den beiden angezeigten Verzeichnissen) kann mittels der `Tab`-Taste hin- und hergewechselt werden. In einem Panel lassen sich Dateien folgendermaßen auswählen:

- Mit den Cursor-Tasten ↑ und ↓ kann in Einzelschritten zwischen den Dateien des Panel-Ordners navigiert werden.
- Mit den `PageUP`- beziehungsweise `PageDown`-Tasten können umfangreiche Ordner seitenweise „durchblättert“ werden.

- Mit der `Home`-Taste gelangt man zum ersten Eintrag eines Verzeichnisses. Dieser ist stets `..` und ermöglicht durch Bestätigung mit der `Enter`-Taste einen Wechsel in das übergeordnete Verzeichnis.
- Mit der `End`-Taste gelangt man zum letzten Eintrag eines Verzeichnisses.
- Mit der `Insert`-Taste können mehrere Dateien ausgewählt oder wieder demarkiert werden. Durch einen Wechsel in ein anderes Verzeichnis wird die aktuelle Auswahl ebenfalls aufgehoben.

Viele häufig auftretende Aktionen lassen sich mittels der folgenden Funktionstasten bewerkstelligen:

`F3`	Ausgewählte Datei(en) mit dem internen Betrachter („Pager“) öffnen.
`F4`	Ausgewählte Datei(en) mit einem Editor öffnen.
`F5`	Ausgewählte Datei(en) vom aktuellen Panel in das gegenüberliegende kopieren.
`F6`	Ausgewählte Datei(en) vom aktuellen Panel in das gegenüberliegende verschieben.
`F7`	Einen neuen Ordner im Verzeichnis des aktuellen Panels erstellen
`F8`	Ausgewählte Datei(en) und/oder Ordner im aktuellen Panel löschen.
`F9`	Menüzeile anwählen

Über das Menü `Optionens` lässt sich der Midnight-Commander bezüglich Aussehen und Verhalten etwas anpassen. Persönlich halte ich folgende Anpassungen für sinnvoll beziehungsweise angenehm:

- Im Bereich `Konfiguration` die Option „Sicheres Löschen“ (mittels der Leertaste) aktivieren, um nicht eine Datei versehentlich durch Drücken der `Del`-Taste, sondern nur mittels $F8$ löschen zu können.
- Im Bereich `Layout` die Menüleiste, Tastenleiste und Informationsleiste (mittels der Leertaste) deaktivieren.
- Im Bereich `Paneloptionen` (ebenfalls mittels der Leertaste) „Lynx-artige Bewegungen erlauben“ aktivieren. Dies erlaubt es, mittels der rechten Cursortaste in das ausgewählte Verzeichnis zu wechseln beziehungsweise mit der linken Cursortaste das übergeordnete Verzeichnis anzuwählen. Dies funktioniert übrigens auch auch mit (komprimierten) Archiv-Dateien!
- Im Bereich `Aussehen` kann ein anderes Farbschema gewählt werden. Persönlich nutze ich am liebsten das zum Standard-Umfang gehörende Theme `modarcon16-efbg-thin`.

Weitere nützliche Tastenkombinationen für die Bedienung des `mc` sind:

`Ins`	Datei unter Cursor markieren beziehungsweise demarkieren
`*`	Alle Dateien des aktuellen Verzeichnisses markieren beziehungsweise demarkieren
`+`	Dateien nach bestimmten Kriterien markieren
`\`	Dateien nach bestimmten Kriterien demarkieren
`Ctrl Leertaste`	Größe des ausgewählten Verzeichnisses anzeigen
`Ctrl s Text`	Im Aktuellen Verzeichnis zu einer Datei springen, die mit `Text` beginnt. Funktioniert auch mit Verzeichnisnamen. Groß-/Kleinschreibung beachten!
`Esc ?`	Im aktuellen Verzeichnis nach Dateien und/oder Inhalten suchen
`Esc Tab`	Auto-Vervollständigung der Eingabezeile (wie `Tab` in einer Shell)
`Esc t`	Zwischen verschiedenen Dateilisten-Layouts wechseln
`Esc ,`	Zwischen horizontaler und vertikaler Fensterteilung wechseln
`Esc .`	Mit `.` beginnende Konfigurationsdateien und -verzeichnisse ein- und ausblenden
`Esc Enter`	Name der Datei unter dem Cursor in die Eingabezeile kopieren
`Ctrl x t`	Die Namen aller markierten Dateien in die Eingabezeile kopieren
`Ctrl x Ctrl t`	Die Namen aller markierten Dateien der anderen Fensterhälfte in die Eingabezeile kopieren
`Ctrl x p`	Den aktuellen Pfadnamen in die Eingabezeile kopieren
`Ctrl x Ctrl p`	Den Pfadnamen der anderen Fensterhälfte in die Eingabezeile kopieren
`Ctrl a Ctrl k`	Die Eingabezeile säubern (an den Anfang gehen und alles bis zum Ende löschen).
`Ctrl O`	Wechsel zwischen Mitnight-Commander und Shell (mit `mc` als Background-Job)
`Ctrl U`	Beide Fensterhälften vertauschen
`Ctrl \`	Verzeichnis-„Hotlist“ anzeigen. Hier lassen sich neben lokalen Pfaden auch FTP- beziehungsweise SSH-Zugangsadressen speichern beziehungsweise aufrufen
`Ctrl x Ctrl s`	Einen Symlink der ausgewählten Datei (beziehungsweise des ausgewählten Ordners) im Pfad der anderen Fensterhälfte erzeugen (anstelle die Datei dorthin zu kopieren)
`Ctrl H`	Liste der zuletzt besuchten Verzeichnisse anzeigen. Auswahl mit Pfeiltasten, Bestätigen mit `Enter`
`Ctrl h`	Liste der letzten Eingabezeilen-Befehle anzeigen. Auswahl mit Pfeiltasten, Bestätigen mit `Enter`

Drückt man `F9` zum Aktivieren der Menüzeile, anschließend unmittelbar `Enter` zum Öffnen der Menüs für das linke beziehungsweise rechte Panel und anschließend `l` für `Listenmodus`, so kann das Aussehen der Panels etwas optimieren, indem man eine benutzerdefinierte Ansicht wählt. Persönlich nutze ich dabei folgende Anzeige:

```
half type name | size:4 | mtime
```

Die Fensterbreite wird damit vorwiegend für die Anzeige der Dateinamen und den jeweiligen Endungen genutzt; am rechten Rand des Panels wird das Datum der letzten Änderung angezeigt (`mtime`); dazwischen wird die Dateigröße eingeblendet, wobei dafür maximal vier Zeichen genutzt werden dürfen. Dadurch wird eine Dateigröße von `3196000` Bytes in Form von `3.2M` dargestellt; es die Dateigröße wird also gerundet und als Kilo- beziehungsweise Mega- oder Gigabyte ausgegeben.

Bisweilen ist es auch nützlich, `F9 Enter s` einzugeben, um in das Menü für das jeweilige Panel zu gelangen und dabei die Sortierreihenfolge festzulegen. Meist mag man die Dateien alphabetisch sortiert aufgelistet haben, bisweilen ist allerdings auch eine Sortierung nach dem Datum der letzten Änderung, nach der Dateigröße oder den Datei-Endungen hilfreich.

mencoder

Das Programm `mencoder` kann für vielerlei Arten von Video-Format-Umwandlungen genutzt werden. Beispielsweise lassen sich damit mehrere Teil-Videos (beispielsweise `.flv`- oder `.mp4`-Dateien von Youtube) folgendermaßen zu einer einzigen Datei zusammenfügen:

```
mencoder -ovc copy -oac copy -o complete-movie.mp4 part1.mp4 part2.mp4
```

Mehr Infos gibt es unter Mencoder im Ubuntuusers-Wiki.

ncdu

Das Programm `ncdu` („ncurses-disk-usage“) ermöglicht es zu sehen, welche Ordner beziehungsweise Dateien am meisten Platz auf der Festplatte benötigen. Ausgehend von dem Pfad, aus dem heraus `ncdu` aufgerufen wird, analysiert es den Speicherbedarf und gibt eine sortierte, navigierbare Verzeichnisliste zurück.

`ncdu` ist über die Paketverwaltung mittels des gleichnamigen Pakets installierbar:

```
sudo aptitude install ncdu
```

Gibt man nach der Installation `ncdu` ein, so wird vom aktuellen Verzeichnis aus die Größe aller sich darin befindenden Dateien bestimmt; die Größe von Unterverzeichnissen wird schrittweise anhand der darin enthaltenen Dateien beziehungsweise Verzeichnisse bestimmt. Als Ergebnis wird der Inhalt des aktuellen Verzeichnisses nach absteigender Größe sortiert aufgelistet.

Eine Navigation innerhalb dieser Übersicht ist mittels den Pfeiltasten oder, wie bei *Vim*, mittels `hjkl` möglich. Drückt man `?`, so wird eine Kurzübersicht der möglichen Eingabetasten eingeblendet, mit `d` kann die Datei unter dem Cursor (nach einer manuellen Bestätigung) gelöscht werden. Mit `q` wird `ncdu` wieder beendet.

nmap

Mit `nmap` können unter anderem anderem die Rechner innerhalb des lokalen Netzwerks mitsamt lokaler Netzwerkadresse aufgelistet werden. Die Syntax hierzu lautet:

```
nmap -sP 192.168.1.0/24
```

In diesem Fall wird davon ausgegangen, dass die zu durchsuchenden Netzwerkadressen die Form `192.168.1.xxx` haben, wobei `xxx` eine laufende Nummer zwischen `1` und `255` sein kann. Ebenfalls möglich sind andere Namensräume, beispielsweise `192.168.2.xxx`. Welcher bei den lokalen Rechnern vorliegt, hängt von den Einstellungen des Routers ab; man kann den lokalen Namensraum mittels Eingabe `ip r` abfragen.

Noch schneller lassen sich die lokalen Rechner samt Netzwerkadressen mittels Eingabe von `arp-scan -l` auflisten; dazu muss lediglich das Programm `arp-scan` mittels `aptitude` installiert werden.

pdfimages

Das `pdfimages`-Programm ermöglicht es, alle in einer PDF-Datei enthaltenen Graphiken beziehungsweise Bilder auf einmal zu „extrahieren“, also als einzelne Bilddateien zu speichern.

`pdfimages` ist Teil des `poppler-utils`-Pakets, das sich folgendermaßen installieren lässt:

```
aptitude install poppler-utils
```

Um alle Bilder zu extrahieren, gibt man im Ordner der PDF-Datei folgendes ein:

```
pdfimages dateiname.pdf dateiname
```

Die Bilder werden dann als `dateiname-001.ppm` usw. gespeichert; mit `pdfimages -f n` beziehungsweise `pdfimages -l n` können jeweils die erste und/oder die letzte zu scannende Seitennummer (`n`) festgelegt werden. Die extrahierten Bilder lassen sich mittels folgendem Skript beispielsweise in PNG-Dateien umwandeln:

```
for i in *.ppm; do convert $i $(basename $i .ppm).png ; done
```

pdftk

Das `pdftk`-Toolkit ermöglicht eine vielseitige Nachbearbeitung von `pdf`-Dateien. In den `man pdftk`-Hilfeseiten finden sich zahlreiche Beispiele, wie man mittels `pdftk` mehrere `pdf`-Dokumente zusammenfügen, einzelne Seiten entfernen, rotieren, oder vertauschen kann.

`pdftk` lässt sich wie üblich aus den Paketquellen installieren:

```
sudo aptitude install pdfk
```

Der grundsätzliche Aufruf erfolgt dann in folgender Form:

```
pdftk inputdatei(en) cat [seitenzahlen] output outputdatei.pdf
```

- Um mehrere pdf-Dokumente zu einer Datei zusammenzufügen, genügt folgender Aufruf:

```
pdftk datei1.pdf datei2.pdf cat output neue-datei.pdf
```

- Um einzelne Seiten aus einer pdf-Datei heraus zu kopieren, kann pdftk folgendermaßen aufgerufen werden:

```
pdftk datei.pdf cat 5 7 9-13 output ausschnitt.pdf
```

Mehr Infos zu pdftk gibt es unter pdftk im Ubuntu-Wiki.

pngnq

Das Programm pngnq kann verwendet werden, um die Dateigröße von Bildern im PNG-Format durch Komprimierung erheblich (teilweise $> 50\%$) zu reduzieren. Dabei werden von pngnq Qualitätsverluste in Kauf genommen, die jedoch mit bloßem Auge (meist) nicht zu erkennen sind – beispielsweise reicht in vielen Fällen zur Darstellung eines Bildes eine Farbtiefe von 256 Farben völlig aus.[3]

Persönlich verwende ich pngnq beispielsweise, um mit Inkscape erstellte und als png exportierte Graphiken zu verkleinern, damit die Webseiten, in denen die Graphiken vorkommen, schneller und mit weniger Serverlast geladen werden können. Dazu nutze ich hintereinander folgende zwei Mini-Skripte:

```
for f in $(find ./ -name '*.png' | grep -v nq8); \
    do pngnq -n 256 $f && rm $f ; done

for f in $(find ./ -name '*.png'); \
    do mv $f $(dirname $f)/$(basename $f | sed 's/-nq8//') ; done
```

Mit der ersten Zeile wird pngnq auf alle png-Dateien angewendet; die neuen Dateien erhalten automatisch die Endung -nq8 angehängt, die Originale werden gelöscht. Im zweiten Schritt werden die neuen Dateien umbenannt, so dass sie wieder mit den ursprünglichen identisch sind (aber im Vergleich zu den Originalen oft nur noch halb so viel Speicherplatz benötigen).

[3] Möchte man keinerlei Qualitätsverlust hinnehmen, so kann das Programm optipng genutzt werden, das via apt als gleichnamiges Paket installierbar ist. Um beispielsweise alle png-Dateien eines Verzeichnisses mit optipng zu optimieren, kann folgender Aufruf genutzt werden:

```
for file in *.png ; do optipng -o9 $file ; done
```

Der Kompressionsgrad von optipng ist allerdings erheblich geringer als von pngnq. Speziell für Webseiten sollte daher vergleichsweise auch pngnq getestet werden, wobei es ratsam ist, von den Originaldateien zunächst eine Backup-Kopie anzufertigen.

screen

Das Programm `screen` ermöglicht es, parallel auf mehreren virtuellen Terminals zu arbeiten, obwohl man in Wirklichkeit nur eines verwendet. Dies kann hilfreich sein, um sich beispielsweise über ssh auf einem externen Rechner einzuloggen und innerhalb des gleichen Fensters mehrere Prozesse ablaufen zu lassen.

`screen` lässt sich wie üblich aus den Paketquellen installieren:

```
sudo aptitude install screen
```

Nach der Installation kann `screen` ohne weitere Der Aufruf von Screen-Funktionen werden allgemein durch Drücken von `Ctrl a` eingeleitet, wobei der Eingabe der darauffolgenden Taste bestimmt, welche screen-Funktion gewählt wird.

Screen legt beim Start ein erstes virtuelles Terminal an. Drückt man hintereinander `Ctrl a` und `c` („create“), so erzeugt screen ein weiteres virtuelles Fenster; die ersten 10 Fenster werden mit Nummer und Bezeichnung in einer Infoleiste am unteren Ende des Screen-Fensters aufgelistet. Die Fenster können über `Ctrl a` und eine Zahl zwischen `0` und `9` ausgewählt werden, weitere können mittels `Ctrl a` und `'` angewählt werden. Mit `Ctrl a` und `"` wird eine Übersicht aller Fenster eingeblendet, die auch zur gezielten Auswahl genutzt werden kann. Eine Änderung des Namens eines Fensters kann mit `Ctrl a` und `a` erreicht werden. Um ein Fenster zu schließen, kann man wie gewohnt `exit` eingeben oder `Ctrl a` und `k` („kill“) drücken.

Anstelle von `screen` kann in den allermeisten Fällen das Nachfolge-Projekt *tmux* genutzt werden, das insbesondere mehr Konfigurations-Möglichkeiten bietet.

tesseract

Mit `tesseract` als Texterkennungs-Software lassen sich im `.tif`-Format eingescannte oder photographierte Texte zurück in Textdateien verwandeln. Im Gegensatz zu den eingescannten Bilddateien sind diese dann durchsuch- und wiederverwertbar.

Installation:

`tesseract` ist über die Paketverwaltung mittels der Pakete `tesseract-ocr` beziehungsweise `tesseract-ocr-deu` und `tesseract-ocr-eng` für deutsche und englische Sprachunterstützung installierbar.

```
sudo aptitude install tesseract-ocr tesseract-ocr-deu tesseract-ocr-eng
```

Auch zahlreiche weitere Sprachen sind verfügbar, die entsprechenden Pakete können mit `aptitude search tesseract` angezeigt werden.

Benutzung:

`tesseract` benötigt als Eingabe-Format `.tif`-Dateien mit maximal acht Graustufen. Der Aufruf von `tesseract` zur Text-Erkennung erfolgt dann für einen deutschsprachigen Text folgendermaßen:

```
tesseract input-file.tif output-file  -l deu
```

Für englischsprachige Dateien wird entsprechend `-l eng` angegeben. An die Ausgabe-Datei wird automatisch die Endung `.txt` angefügt.

Tip: Screenshots nutzen

Neben der Limitierung auf acht Graustufen hat `tesseract` den Nachteil, dass der Original-Text einspaltig sein sollte – eine Trennlinie oder Tabulatur zwischen mehreren Spalten wird schlichtweg ignoriert. Wenn eine mehrspaltiger oder mehrfarbig gescannte beziehungsweise photographierte Original-Datei vorliegt – womöglich noch dazu in einem anderen Dateiformat –, so kann man sich, falls man sich ohnehin nur für bestimmte Ausschnitte interessiert, mit einem entsprechenden Screenshot-Alias in der `~/.bashrc` helfen:

```
alias it='import -depth 8 txt_$(date +%Y%m%d_%H%M%S).tif'
```

Persönlich verwende ich das obige Beispiel zur Aufnahme von Screenshots mittels des `import`-Befehls aus dem Imagemagick-Paket. Die Angabe `-depth 8` legt die Anzahl der Graustufen des Screenshots auf acht fest. Die Namen der einzelnen Screenshots sollen dann einem einheitlichen Namensmuster und schließlich chronologisch sortiert vorliegen; dies wird durch die Nutzung des `date`-Befehls erreicht.

Wird die obige Code-Zeile in die Konfigurationsdatei `~/.bashrc` kopiert (und diese im gleichen Shell-Fenster gegebenenfalls mit `source ~/.bashrc` neu geladen), so kann mittels Eingabe von `it` stets ein neuer Screenshot im aktuellen Verzeichnis gespeichert werden – es muss nur noch mit dem erscheinenden Fadenkreuz ein Bidschirm-Bereich für den Screenshot festgelegt und durch einen Mausklick bestätigt werden.

Tip: Stapelverarbeitung mehrerer .tif-Dateien:

Mag man mehrere `.tif`-Dateien auf einmal der Texterkennung zuführen, so kann dies mit folgendem Einzeiler-Skript erreicht werden:

```
for i in *.tif; do tesseract $i $i -l deu; done;
```

Die Ausgabedateien werden in diesem Fall nach den Eingabe-Dateien benannt, gefolgt von der automatischen Endung `.txt`.

Um auf diese Weise zusammengehörige Screenshots eines Buchs der Texterkennung zuzuführen und die erzeugten Dateien wieder zu vereinen, sollten die Screenshots zum einen in einem separaten Ordner aufgenommen beziehungsweise dorthin kopiert werden. Zum anderen sollten die Screenshots entlang eines Buches stets „von vorne nach hinten“ aufgenommen werden, da auch die resultierenden Bild- beziehungsweise Textdateien chronologisch sortiert sind.

Das Zusammenführen aller Textdateien eines Ordners zu einer neuen Zieldatei gelingt schließlich folgendermaßen:

```
for in in *.txt; do cat $i >> new-file.rst ; done
```

Für die Zieldatei nutze ich gerne die Endung `.rst`, einerseits, um bei möglichen späteren Erweiterungen Namenskonflikte zu vermeiden (hierbei würde eine Zieldatei `new-file.txt`

bei einem erneuten Aufruf des obigen Befehls mit auf sich selbst abgebildet werden), andererseits, um den Text gleich für eine „Informationsverwaltung“ oder spätere Publikationen mittels Sphinx bereit zu halten.

Der letztliche „Workflow“ sieht möglicherweise so aus: Farbig gescannte PDF-Datei mit `atril`, `evince` oder einem anderen Dokumentenbetrachter öffnen; in einem Shell-Fenster (beispielsweise *guake*) zu einem gewünschten Zielordner wechseln; mittels Eingabe von `it` und Textauswahl mit der Maus wiederholt Screenshots (beliebig viele) erzeugen; die obigen beiden Einzeiler ausführen und fertig!

Bei einigermaßen guten Scans und einer brauchbaren Auflösung des Bildschirms beim Erzeugen der Screenshots – hierbei genügt eine Vollbild-Darstellung des Dokuments auf einem 17-Zoll-Monitoren in den allermeisten Fällen, bei kleinen Schriftgrößen oder kleineren Monitoren notfalls etwas „hineinzoomen“ und lieber mehrere kleinere Bildausschnitte wählen – sollte das Ergebnis von `tesseract` durchaus zufriedenstellend sein.

Tip: Tiff-Dateien nachbearbeiten

Möchte man mehrere `.tif`-Dateien zu einer Multipage-Tiff-Datei zusammenfügen oder einzelne Seiten einer Multipage-Tiff-Datei entfernen, bieten sich die Hilfsprogramme `tiffcp`, `tiffsplit` und `tiffcrop` an.

Diese können mittels des `libtiff-tools`-Paketes installiert werden:

```
sudo aptitude install libtiff-tools
```

Das gleiche Paket stellt auch den Befehl `tiff2pdf` zur Umwandlung einer Multipage-Tiff-Datei in ein PDF-Dokument bereit.

Schließlich kann auch das Programm `unpaper` zur Aufbesserung von Scans genutzt werden. Infos hierzu gibt es unter unpaper im Ubuntuuser-Wiki.

whois

Mit `whois` können Informationen über den Betreiber, den Standort und das System des Servers angezeigt werden, der zu einer bestimmten IP-Adresse gehört; letztere kann zuvor mittels `host` ermittelt werden:

```
host grund-wissen.de
# grund-wissen.de has address 188.40.57.88
# grund-wissen.de mail is handled by 10 mail.grund-wissen.de.

whois 188.40.57.88
# ... viele Infos ...
```

Üblicherweise befindet sich unter den angezeigten Informationen auch eine Email-Adresse des Server-Administrators.

Administrator-Programme

Als Super-User (manchmal auch „Root“ genannt) kann man neben den *Standard-Programmen* und möglichen zusätzlich installierten *Zusatz-Programmen* auch Werkzeuge nutzen, die systemweite Veränderungen bewirken können. Im Folgenden ist eine kleine Auswahl dieser Programme aufgelistet:

adduser, deluser

Mit `adduser benutzername` wird ein neues Benutzer-Konto erstellt und der entsprechende Ordner im `/home`-Verzeichnis angelegt. Mit `deluser benutzername` kann entsprechend ein bestehendes Benutzer-Konto gelöscht werden. Standardmäßig bleibt dabei das Home-Verzeichnis des Benutzers erhalten; möchte man dieses gleich mit entfernen, kann `deluser` mit der Option `--remove-home benutzername` aufgerufen werden.

apt, aptitude

Mit `apt-cache search ...` (oder einfacher: `aptitude search ...`) kann jeder Benutzer die lokale Pakete-Liste nach dem Namen oder der Beschreibung eines Programms oder einer Code-Bibliothek (lib) durchsuchen (und somit beispielsweise nachsehen, ob ein bestimmtes Paket installierbar beziehungsweise installiert ist).

Als Super-User kann man zusätzlich weitere Funktionen von `apt` beziehungsweise `aptitude` nutzen:[1]

- Mit `apt-get update` (oder einfacher: `aptitude update`) wird die lokale Pakete-Liste aktualisiert; damit verbunden wird auch die Liste der Server, von denen die Pakete heruntergeladen werden können, auf den aktuellen Stand gebracht. Ein solcher Update sollte in regelmäßigen Abständen beziehungsweise vor Paket-Installationen und System-Upgrades vorgenommen werden.
- Mit `apt-get install paketname` (oder einfacher: `aptitude install paketname`) lässt sich ein Programm oder eine Bibliothek aus der Paketliste installieren; es können auch mehrere Paketnamen auf einmal angegeben werden: `aptitude install paket1 paket2 ...`.

  Führt eine Installation zu Versions-Konflikten, sucht `apt` beziehungsweise `aptitude` automatisch nach einer Lösung, welche die verletzten Abhängigkeiten nach Möglichkeit auflöst; es erscheint eine entsprechende Programm-Rückmeldung, wobei es dem Benutzer überlassen wird, ob die Lösung akzeptiert wird oder ob weiter nach einer anderen Lösung gesucht werden soll.

[1] In der Datei `/etc/apt/sources.list` ist festgelegt, auf welchen Servern `apt` nach verfügbarer Software suchen beziehungsweise diese installieren soll. Je nach Vorlieben können stets die aktuellsten Entwicklungen oder nur ältere, bereits bewährte Pakete abgefragt werden.

Durch Aufruf von `apt-get update` wird die Liste an verfügbaren Paketen aktualisiert. Zusätzlich zu jeder Quelle, die durch einen `deb`-Eintrag festgelegt wird, kann ein `deb-src`-Eintrag stehen, wenn von dort auch Quellcode heruntergeladen werden soll (interessant für Entwickler beziehungsweise um Programme selbst zu kompilieren).

- Mit `aptitude reinstall paket` kann ein einzelnes Paket (Programm oder Bibliothek) neu installiert werden. Dabei wird gegebenenfalls automatisch ein Update vorgenommen. Mögliche Konfigurationsdateien bleiben dabei unberührt.
- Mit `apt-get remove paket` (oder einfacher: `aptitude remove paket` kann ein installiertes Paket wieder deinstalliert werden. Die Konfigurationsdateien bleiben dabei erhalten; um auch diese zu entfernen, kann der Aufruf `apt-get purge paket` beziehungsweise `aptitude purge paket` genutzt werden.
- Mit `apt-get autoclean` werden alle noch teilweise installierten Pakete vom Rechner entfernt, die von den aktuell installierten Programmen nicht mehr genutzt werden.
- Mit `apt-get source paket` kann der Quellcode eines Pakets (Programm oder Bibliothek) herunter geladen werden. Somit kann man das Programm selbst compilieren beziehungsweise bei entsprechenden Programmier-Kenntnissen einen Blick in die eigentliche Funktionalität des Programms werfen und/oder Teile des Quellcodes für eigene Projekte weiter verwenden.
- Mit `apt-get upgrade` beziehungsweise `aptitude safe-upgrade` werden alle möglichst alle Pakete auf den neuesten Stand gebracht; ein Installieren zusätzlicher Pakete und/oder ein Entfernen bestehender Pakete soll dabei bei verletzten Abhängigkeiten (möglichst) ausbleiben. Es erscheint eine Rückmeldung über die anstehenden Aktualisierungs-Maßnahmen, wobei sich der Benutzer entscheiden kann, ob die angebotene Lösung akzeptabel ist oder nach einer anderen Lösung gesucht werden soll.

  Vor einem Upgrade sollte die lokale Paket-Liste stets mit `apt-cach upgrade` beziehungsweise `aptitude update` aktualisiert werden.

  Mit `aptitude full-upgrade` besteht darüber hinaus bei Versions-Konflikten die Möglichkeit, die Aktualisierung der installierten Pakete durch eine Installation weiterer Pakete und/oder ein Entfernen bestehender Pakete zu „erleichtern“. Hier muss von Fall zu Fall geprüft werden, ob ein solcher Upgrade den eigenen Wünschen/Erwartungen entspricht.
- Mit `apt-get dist-upgrade` beziehungsweise `aptitude dist-upgrade` werden nicht nur einzelne Pakete, sondern gegebenenfalls die gesamte Distribution aktualisiert (beispielsweise ein Wechsel von Ubuntu Version 12.04 auf Version 12.10). Da ein solcher System-Upgrade zahlreiche Veränderungen mit sich bringen kann, sollte man sich den Einsatz dieser Aktualisierungs-Variante vorher gut überlegen.[2]

`aptitude` kann auch ohne zusätzliche Kommandozeilen-Argumente aufgerufen werden. In diesem Fall erscheint eine text-basierte Benutzeroberfläche. In der oberen Hälfte werden Paketnamen (nach verschienenen Rubriken sortiert) aufgelistet, in der unteren Hälfte erscheint zum jeweiligen Paket eine passende Beschreibung. Mit den Cursor-Tasten kann man sich in der oberen Bildschirmhälfte durch die Paketzweige navigieren, mit der `Enter`-Taste werden Unterkategorien ein- beziehungsweise ausgeblendet. Wie in den obersten Zeilen kurz beschrieben, kann mit `q` das Programm beendet werden; mit `u` werden die Paketquellen aktualisiert (entspricht `aptitude update`). Mit `+` kann das Paket unter dem

[2] Tip für private Desktop-PCs: Zwei lauffähige Linux-Varianten parallel installieren! So kann das weniger genutzte System als „Experimentier-Umgebung“ genutzt werden.

Cursor zur (Neu-)Installation, mit - zum Entfernen und mit _ zum vollständigen Löschen vorgemerkt werden; mit g werden die vorgemerkten Änderungswünsche ausgeführt.

chmod

Mit `chmod` kann festgelegt werden, welche Dateirechte (Lesen, Schreiben, Ausführen) für den Eigentümer, für die Benutzer-Gruppe und für alle Anderen gelten.

chown, chgrp

Mit `chown benutzername dateien` kann der Eigentümer einer oder mehrerer Dateien festgelegt werden; entsprechend können mit `chgrp gruppenname dateien` eine oder mehrere Datei(en) einer Benutzer-Gruppe zugewiesen werden. Mit der Option `-R` lassen sich beide Werkzeuge auch rekursiv auf Verzeichnisse mitsamt Unterverzeichnissen anwenden.

chroot

Mit `chroot pfad` kann der angegebene Pfad als Basispfad / des Betriebsystems festgelegt werden. Dies ist insbesondere praktisch, um ein „Live-System“ von einem USB-Stick zu booten und von diesem System aus Wartungen am eigentlich installierten Betriebsystem vorzunehmen (falls dieses aus irgendwelchen Gründen nicht mehr booten sollte). Nützlich ist dabei folgende Routine:[3]

```
# Vorab: Die System-Partition eingebinden:
# -- falls unbekannt: fdisk -l eingeben! --
# mount /dev/[systempartition] pfad

cd pfad mount --bind /sys ./sys mount --bind /dev ./dev mount --bind /proc
./proc chroot .
```

Hierbei werden (nach einem Wechsel in den Pfad der Systempaftition) zunächst die Systemdaten (abgelegt in `/sys`), die Daten der angeschlossenen Geräte (abgelegt in `/dev`) und der laufenden Prozesse (abgelegt in `/proc`) in das zu wartende System eingebunden. Anschließend kann man mit `chroot .` das entsprechende Verzeichnis als Basispfad nutzen und somit innerhalb des Shell-Fensters Programme direkt auf dem zu wartenden System ausführen. Mit `exit` kann man in das eigentlich laufende (Live-)System zurück gelangen.

dpkg

Der Debian Package Manager (`dpkg`) ist die Basis-Anwendung zum Installieren und Deinstallieren von Debian-Paketen; auch `apt` macht intern von `dpkg` Gebrauch. Auch wenn ein Programm nicht in den Paket-Quellen enthalten ist, kann es von einer entsprechenden Webseite (beispielsweise von Sourceforge) heruntergeladen und Download-Ordner mit `sudo dpkg -i paket.deb` installiert werden. Entsprechend kann es mit `sudo dpkg -r`

[3] Mit . wird der Pfad des aktuellen Verzeichnisses bezeichnet.

`paketname` wieder (unter Beibehaltung der Konfigurationsdateien) deinstalliert oder mit `dpkg -P paketname` restlos entfernt werden.

Mit `dpkt -l suchbegriff` lassen sich alle Pakete auflisten, die auf einen Suchbegriff zutreffen – reguläre Ausdrücke können ebenfalls eingesetzt werden. Mit `dpkg -S paketname` wird angezeigt, welche Datei(en) durch das entsprechende Paket installiert wurden.

eject

Mit `eject devicename`, beispielsweise `eject /dev/cdrom0`, kann das CD/DVD-Laufwerk geöffnet werden.

fdisk

Mit `fdisk` können Informationen über interne und externe Festplatten beziehungsweise Speichermedien angezeigt werden. Ebenso ist es möglich, mit `fdisk` Partitionen zu verwalten.

Nützlich ist der Aufruf von `fdisk -l`, um Disk-Informationen angeschlossener Speichermedien anzuzeigen.

halt, reboot

- Mit `halt` kann das System herunter gefahren und der Computer ausgeschaltet werden.
- Mit `reboot` kann das System neu gestartet werden.

iftop

Mit `iftop` kann der Datenaustausch angezeigt werden, der zwischen dem eigenen und anderen Rechnern stattfindet. Das Programm kann folgendermaßen installiert werden:

```
sudo aptitude install iftop
```

Anschließend kann es in einer Shell mittels `iftop` gestartet werden. Die Anzeige ist interaktiv (ebenso wie bei *top*) und kann durch Drücken von `q` wieder beendet werden. Drückt man `h`, so bekommt man eine kurze Hilfe angezeigt, wie sich das Programm durch Drücken einzelner Tasten steuern lässt.

lshw

Mit `lshw` werden die Hardware-Informationen des Computers aufgelistet; mit `lshw -short` wird eine Kurzform dieser Informationen ausgegeben.

mount, umount

Mit `mount device pfad` kann ein Datenträger (Speichermedium, Partition oder Ordner) in den angegebenen Pfad einbinden („mounten“); entsprechend wird mit `umount pfad` die Einbindung gelöst, falls kein Programm aktuell auf das im angegebenen Pfad eingebundene Medium zugreift.

Ruft man `mount` ohne weitere Argumente auf, so bekommt man eine Liste mit den verschiedenen Devices, ihren Mount-Points, Lese- beziehungsweise Schreib-Optionen und Datei-Systemen angezeigt; diese Liste kann durch übersichtlicher `mount | column -t` in einer übersichtlicheren Form ausgegeben werden.

nast

Das Programm `nast` ist nicht standardmäßig auf jedem Linux-System installiert, kann aber einfach nachinstalliert werden:

```
sudo aptitude install nast
```

Mit `nast -m` können dann die IP-Adressen aller Rechner und Router, die sich im lokelen Netzwerk befinden, aufgelistet werden.

su

Mit `su benutzername` kann sich ein Superuser als beliebiger anderer Benutzer anmelden. Mit `sudo su root` kann sich ein beliebiger Benutzer, der dazu berechtigt ist, sich in einer Shell dauerhaft Superuser-Rechte verschaffen; dazu muss ein entsprechender Eintrag folgender Form in der Datei `/etc/sudoers` existieren:

```
# User privilege specification
benutzername        ALL=(ALL:ALL) ALL
```

Der Benutzerwechsel kann mit `exit` wieder beendet werden.

usermod

Mit `usermod` kann das Konto eines Benutzers verändert werden. Beispielsweise kann mittels `usermod -aG gruppenname benutzername` ein Benutzer zur angegebenen Gruppe hinzugefügt werden; dies ist unter anderem nützlich, um als normaler Benutzer einen Arduino über die serielle Schnittstelle ansprechen zu können.

Shell-Scripting

Die Shell kann unmittelbar als Interpreter für einzelne Programmaufrufe genutzt werden. Darüber hinaus ist es ebenso möglich, mehrere Eingabezeilen miteinander zu kombinieren.

Diese Methode, mehrere aufeinander folgende Programmaufrufe – ähnlich wie ein Kochrezept – in eine Textdatei zu schreiben und mittels dieser Datei von der Shell ausführen zu lassen, wird als „Shell-Scripting“ bezeichnet.

In den folgenden Abschnitten werden einige Möglichkeiten zum gezielten Scripten der `bash`, also der unter Debian, Ubuntu und Linux Mint am weitesten verbreiteten Shell, näher vorgestellt.

Aufbau und Aufruf eines Shellskripts

Shell-Skripte sind Textdateien, die üblicherweise die Endung `.sh` oder überhaupt keine Endung besitzen.[1] In beiden Fällen sollte am Anfang der Datei festgelegt werden, welcher Shell-Interpreter beim Aufruf des Skripts genutzt wird. Dies erfolgt über den so genannten „Shebang“:[2]

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

Die Zeichenkette `#!` bewirkt dabei, dass die nachfolgende Anweisung (gegebenenfalls mit zusätzlichen Argumenten) beim Aufruf des Programms ausgeführt wird. Im obigen Beispiel wird also ein `bash`-Prozess gestartet, der die darauf folgenden Anweisungen und Programmaufrufe Zeile für Zeile interpretiert.[3] Alle anderen Zeilen, die mit `#` beginnen, werden als Kommentare ignoriert.

Um das Shell-Skript aufzurufen, wird in einer Shell folgende Zeile eingegeben:

```
sh pfad/skriptdatei
```

Ebenso ist es möglich, die Skriptdatei mit `chmod +x` ausführbar zu machen. Dann kann sie – wie ein binäres Programm – direkt aufgerufen werden:

```
# Datei ausführbar machen:
cd pfad
chmod +x skriptdatei

# Skript aufrufen:
pfad/skriptdatei
```

Befindet man sich im gleichen Ordner wie die Skriptdatei, so kann diese mit `./skriptdatei` aufgerufen werden, da `.` für den Namen des aktuellen Verzeichnisses steht.

Möchte man bestimmte Skripte auch ohne explizite Angabe des Pfades aufrufen, so empfiehlt es sich zu diesem Zweck mit `mkdir ~/bin` ein eigenes Unterverzeichnis im Home-

[1] Die Dateinamen von Shell-Skripten sollten keine Zeichen außer Groß- und Kleinbuchstaben, Nummern und dem Unterstrich beinhalten; Umlaute und Sonderzeichen sollten, obwohl sie prinzipiell zulässig sind, vermieden werden.

[2] Auf die gleiche Weise kann man zu Beginn einer Skriptdatei auch einen anderen Interpreter festlegen. Beispielsweise leiten `#!/bin/awk -f` ein AWK-Skript oder `/usr/bin/python3` ein Python3 -Skript ein.

[3] Möchte man in eine Zeile zwei oder mehrere Anweisungen schreiben, so müssen diese durch `;` getrennt werden. (Andernfalls würde die zweite Anweisung als Argument der ersten Anweisung interpretiert werden.)

Verzeichnis anzulegen und in die Konfigurationsdatei `~/.bashrc` folgenden Eintrag zu schreiben:

```
# Eigenes bin-Verzeichnis zum Systempfad PATH hinzufügen

# Prüfen ob das Verzeichnis existiert:
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
    export PATH;
fi
```

Durch die Zeile `PATH="$HOME/bin:$PATH"` wird das eigene `bin`-Verzeichnis an den Anfang des Systempfades gesetzt. Wird in einer Shell eine beliebige Anweisung eingegeben, so wird zunächst der eigene `bin`-Ordner und erst anschließend alle anderen in der `PATH`-Variable gespeicherten Verzeichnisse nach einem entsprechenden Programm beziehungsweise Skript durchsucht.

Rückgabewerte und Verkettung von Programmen

Jedes Programm oder Shell-Skript sollte beim Beenden einen Wert liefern, der angibt, ob das Skript fehlerfrei abgelaufen ist oder nicht. Üblicherweise wird dazu das Programm bei einem fehlerfreien Durchlauf mit `exit 0` beendet. Jeder andere „Exit-Status“ zwischen `1` bis `255` deutet meistens auf einen Fehler hin.[4] Der Rückgabewert wird in einem Skript an einer beliebigen Stelle mittels `exit num` festgelegt; das Skript wird dadurch unmittelbar beendet.

Bei der interaktiven Benutzung der Shell wird der Exit-Status eines Skripts oder Programms nur selten benutzt, da mögliche Fehlermeldungen direkt vom Benutzer abgelesen werden können. In Shell-Skripten jedoch werden bestimmte Aktionen häufig in Abhängigkeit von anderen Aktionen ausgeführt; sollen beispielsweise Dateien verschoben werden, so dürfen die Originale nur dann gelöscht werden, wenn sie zuvor erfolgreich kopiert worden sind.

Die Operatoren `&&` und `||`

Mittels der Operatoren `&&` und `||` kann man die Ausführung einer zweiten Anweisung vom Rückgabewert der ersten Anweisung abhängig machen:

- Ein Ausdruck der Form `anweisung1 && anweisung2` bedeutet, dass `anweisung2` nur dann ausgeführt wird, wenn `anweisung1` fehlerfrei ausgeführt beziehungsweise mit Exit-Status `0` beendet wurde.
- Ein Ausdruck der Form `anweisung1 || anweisung2` bedeutet, dass `anweisung2` genau dann ausgeführt wird, wenn `anweisung1` mit einem Fehler beziehungsweise mit einem Exit-Status zwischen `1` und `255` beendet wurde.

Bedingte Anweisungen können auch mittels `if` und `case` implementiert werden.

[4] Wird ein Shell-Skript nicht explizit mittels `exit` beendet, so entspricht der Exit-Status dem Rückgabewert der zuletzt ausgeführten Anweisung.

Ausgaben in Textdateien umlenken

Normalerweise geben Skripte und Programme Meldungen und Rückgabewerte als Text im Shell-Fenster aus. Mit den Operatoren `>` und `>>` ist es allerdings auch möglich, die Ausgabe in Text-Dateien umzuleiten:

- Mit dem Operator `>` kann man die Ausgabe eines Programms in eine Datei umleiten, deren Name im Anschluss an das `>`-Zeichen geschrieben wird:

  ```
  # Inhalt des Verzeichnisses und aller Unterverzeichnisse
  # in Text-Datei "folder-content.txt" schreiben:
  ls -R > folder-content.txt
  ```

  Existiert die Datei noch nicht, so wird sie neu angelegt; als Eigentümer der Datei wird dabei der Benutzer eingetragen, der den Shell-Prozess ausführt.

  Existiert die Datei schon, so wird sie zunächst geleert und anschließend neu beschrieben. Der Eigentümer und die Zugriffsrechte bleiben dabei erhalten. Damit das Überschreiben funktioniert, muss das Schreiben der Datei erlaubt sein.

- Nach dem gleichen Prinzip kann man mit dem Operator `>>` die Ausgabe eines Programms an eine Datei anhängen. Diese Variante findet insbesondere bei der Protokollierung einzelner Prozesse in Log-Dateien Anwendung.

Umgekehrt kann man mittels des Operators `<` eine Datei angeben, aus der gelesen werden soll. Selten wird diese Syntax von Programmen zwingend gefordert, in den meisten Fällen kann eine einzulesende Datei auch ohne `<` angegeben werden. Beispielsweise sind die Anweisungen `cat anyfile.txt` und `cat < anyfile.txt` identisch.

Fehlermeldungen umlenken

Für jeden Prozess öffnet die Shell drei Standard-Kanäle: Kanal `0` steht für die Standard-Eingabe, Kanal `1` für die Standard-Ausgabe und `2` für den Standard-Fehlerkanal.

Man kann jedem der Umleitungskommandos `<`, `>` und `>>` eine dieser Nummern für den jeweiligen Kanal voranstellen. So kann beispielsweise die Fehlerausgabe einer Anweisung mittels `2> error-logfile.txt` in eine entsprechende Log-Datei umgelenkt werden.

Als eine Besonderheit ist hierbei die Datei `/dev/null` hervorzuheben. Diese Datei dient als „Mülleimer“, es werden also alle Meldungen, die zu dieser Datei umgelenkt werden, verworfen. So kann man beispielsweise mittels der Anweisung `any_program 2> /dev/null` die Ausgabe von Fehlern unterdrücken.

Pipelines („Pipes“)

Die Ausgaben eines Programms können nicht nur Dateien, sondern auch an andere Programme weitergeleitet werden. Hierzu wird in Shell-Skripts der Operator `|` („Pipe“) in der Form `anweisung1 | anweisung2` verwendet:

```
# Alle Dateien des aktuellen Verzeichnisses und aller
# Unterverzeichnisse anzeigen, die "txt" enthalten:
ls -R | grep txt
```

Bei einer solchen Verkettung von Programmen werden die Daten vom Interpreter in eine temporäre Datei (ebenfalls „Pipe“ genannt) abgelegt und so an das nächste Programm übergeben.

Pipes stellen ein vielseitiges Werkzeug dar, insbesondere in Kombination mit folgenden Programmen:

- Mit *grep* kann die Ausgabe eines Programms hinsichtlich bestimmter Suchmuster gefiltert werden.
- Mit *tee* kann die Standard-Ausgabe oder Standard-Fehlerausgabe sowohl auf dem Bildschirm ausgegeben als auch in eine Datei geschrieben werden. Die Syntax hierzu könnte also `anweisung | tee error-logfile.txt` lauten.
- Mit `xargs` werden alle empfangenen Werte als Argumente der folgenden Anweisung übergeben. Beispielsweise würde die Anweisung `find ./ -name *foo* | xargs grep muster` alle Dateien, die „foo“ in ihrem Dateinamen enthalten, nach dem gegebenen Begriff oder Suchmuster durchsuchen (ohne `xargs` würden hingegen die Dateinamen nach `muster` durchsucht).

Dateimuster und Variablen

Die Shell weist als Interpreter einigen Sonderzeichen eine besondere Bedeutung zu. Mit Hilfe solcher Zeichen (so genannten „Wildcards“) können Muster für Dateinamen einfach formuliert werden. Die Shell ersetzt dann bei der Ausführung die Muster dann durch die entsprechenden Dateinamen.

Zeichen	Bedeutung
`~`	Der Pfad des Home-Verzeichnisses des aktuellen Benutzers
`*`	Eine beliebig lange Folge von Zeichen
`?`	Ein einzelnes beliebiges Zeichen
`/`	Trennzeichen für Verzeichnis-Namen (Pfadangaben)
`[abc123]`	Eines der Zeichen, die in der Klammer vorkommen
`[a-z1-9]`	Eines der Zeichen im angegebenen Bereich
`[!abc]`	Ein beliebiges Zeichen, das *nicht* in der Klammer vorkommt

Wohl am häufigsten wird das `*`-Zeichen verwendet, das für eine beliebig lange Folge von Zeichen steht; dabei ist auch die Länge Null explizit erlaubt. Beispielsweise werden mittels `ls *foo*` alle Dateien ausgegeben, die „foo“ in ihrem Dateinamen beinhalten, egal welche Zeichen vorher oder nachher im Dateinamen vorkommen. Mit `ls *.txt` werden alle Dateien angezeigt, deren Dateiname auf „.txt“ endet. Zu beachten ist hierbei jedoch eine Ausnahme: Der Stern als Suchmuster ignoriert Dateien, deren Name mit einem Punkt beginnt, es sei denn, man schreibt explizit `.*txt`. Dadurch soll verhindert werden, dass versehentlich Konfigurationsdateien gelöscht werden.

In den eckigen Klammern kann auch ein Buchstaben- oder Zahlenbereich in der Form `[a-z]` oder `[0-9]` angegeben werden; auch eine Kombination der Form `[a-zA-Z0-9]` ist möglich, um ein beliebiges alphanumerisches Zeichen auszudrücken. Diese Syntax funktioniert ebenso für ausschließende Klammern, beispielsweise steht `[!a-z]` für ein beliebiges Zeichen außer einem Kleinbuchstaben.

Eine weitere besondere Bedeutung hat das Dollar-Zeichen `$`: Es ersetzt den unmittelbar (ohne Leerstelle) folgenden Variablennamen durch den in der Variablen abgespeicherten Wert.

Möchte man die gewöhnliche Bedeutung eines Zeichens aufheben, so muss diesem das Backslash-Zeichen \ vorangestellt werden. Dies betrifft sowohl die oben angegebenen Sonderzeichen wie auch das Leerzeichen, das eigentlich zur Trennung verschiedener Argumente genutzt wird, aber auch Bestandteil von Dateinamen sein kann. In gleicher Weise muss den Zeichen `;` `&` `|` `(` `)` `<` `>` sowie `\n` und `\t` (Zeilenende und Tabulator) ein Backslash vorangestellt werden, um die jeweilige Sonderbedeutung aufzuheben.

Eine weitere bisweilen nützliche Ergänzung bieten geschweifte Klammern innerhalb von Dateimustern. Diese eignen sich dazu, um an der gegebenen Stelle einen der in den geschweiften Klammern stehenden, durch Komma-Zeichen voneinander getrennten Namen einzusetzen. Beispielsweise gibt `ls -R ~/data/{buecher,docs}/?*/{*.pdf,*djvu}` alle `pdf`- und `djvu`-Dateien in den Unterverzeichnissen von `~/data/buecher` und `~/data/docs` auf. Diese Liste kann dann beispielsweise mittels `| grep` gezielt nach Einträgen durchsucht werden.

Zuweisung von Variablen

Ähnlich wie in Programmiersprachen, so lassen sich auch in der Shell Werte in Variablen speichern. Allerdings sind nur Zeichenketten („Strings“) als Werte erlaubt.

Um einer Variablen einen Wert zuzuweisen, muss folgende Syntax verwendet werden:

```
variablenname=wert
```

Zwischen dem Variablennamen, dem Zuweisungsoperator = und dem zu speichernden Wert darf dabei kein Leerzeichen stehen; auf den Inhalt der Variable kann wiederum mittels `$variablenname` zugegriffen werden.

Mit der Anweisung `set` kann in der Shell abgefragt werden, welche Variablen aktuell gesetzt sind und welche Werte diese haben. Unter Umständen kann diese Liste recht lang sein, so dass es nützlich ist, die Ausgabe von `set` mittels einer Pipe entweder an einen Pager wie *less* zu übergeben oder mittels *grep* nach einem bestimmten Variablennamen zu filtern.

```
# Alle Variablen und ihre Werte mit less betrachten:
set | less

# Wert der Variablen EDITOR prüfen:
set | grep EDITOR
```

Variablen können in der Shell an jeder beliebigen Stelle genutzt werden. Trifft der Shell-Interpreter auf ein `$`-Zeichen, so wird der unmittelbar (ohne Leerzeichen) folgende Variablenname durch den gespeicherten Variablenwert ersetzt. Ist die angegebene Variable nicht definiert, so wird vom Interpreter an dieser Stelle nichts eingesetzt.

Mittels `unset variablenname` kann man eine Variable wieder löschen.

Exportieren von Variablen

Weist man in einer Shell einer Variablen einen Wert zu, so ist diese Variable per Voreinstellung nur dem aktuellen Shell-Prozess bekannt. Möchte man eine Variable auch in von der aktuellen Shell-Sitzung aus gestarteten Unterprozessen nutzen, so kann sie mittels der `export`-Anweisung zugänglich gemacht werden:

```
# Variable definieren:
my_var=fooo

# Variable öffentlich machen:
export my_var
```

Auch in diesem Fall kann die Variable mittels `unset my_var` wieder gelöscht werden. Wird dem gleichen Variablennamen erneut ein Wert zugewiesen, so wird die Variable wieder als lokal angesehen und muss bei Bedarf erneut exportiert werden.

Definition von Konstanten

Mittels der Anweisung `readonly variablenname` kann eine Variable in eine Konstante umgewandelt werden. Der Wert, den die Variable zu diesem Zeitpunkt hat, kann später nicht mehr verändert werden, auch kann die Variable nicht mehr mittels `unset` gelöscht werden – sie ist sozusagen schreibgeschützt. Erst mit dem Beenden der Shell wird die Konstante wieder gelöscht.

Mittels `readonly` (ohne Variablennamen) kann eine Liste mit allen aktuell definierten Konstanten ausgegeben werden.

Definition von Variablen-Listen

In einer Shell-Variable kann eine Liste mit mehreren Elementen mittels folgender Syntax abgespeichert werden:

```
var_liste=(
    element_1
    element_2
    element_3
)
```

Wichtig ist hierbei wiederum, dass zwischen dem Namen der Variablenliste, dem Zuweisungsoperator `=` und der öffnenden Klammer *kein* Leerzeichen stehen darf. Es können

auch mehrere Elemente in eine Zeile geschrieben werden; die Elemente werden durch Leerzeichen getrennt.

Auf die einzelnen Elemente der Variablenliste kann mittels `${var_liste[0]}`, `${var_liste[1]}` usw. zugegriffen werden, wobei der Index `0` für das erste Listenelement und der Index `1` für das zweite Listenelement steht. Schreibt man nur `$var_liste`, so ist dies mit einem Zugriff auf das erste Element der Liste identisch. Um alle Listenelemente auszugeben, muss hingegen `${var_liste[*]}` geschrieben werden.

Die Anzahl an Elementen einer Liste kann mittels `${#var_liste[*]}` ausgegeben werden; mit `${#var_liste[num]}` wird ausgegeben, aus wie vielen (Text-)Zeichen das Listenelement mit der Indexnummer `num` besteht.

Ein neues Element kann folgendermaßen an eine bestehende Liste angefügt werden:

```
var_liste+=( element_4 )
```

Soll ein neues Element nicht am Ende der Liste, sondern vor einer bestimmten Indexposition `num` eingefügt werden, so kann man folgendes schreiben:

```
var_liste[num]+=( element_neu )
```

Mittels `unset var_liste` kann die Variablenliste, mit `unset var_liste[num]` das Listenelement mit der Indexnummer `num` wieder gelöscht werden.

Besondere Shell-Variablen

Im folgenden werden einige Standard-Variablen aufgelistet, die automatisch definiert sind und häufig in Shell-Skripten vorkommen:

`$0`	Diese Variable enthält den Namen des aktuellen Prozesses, beispielsweise `/bin/bash`. Im Fall eines laufenden Shellskripts entspricht `$0` dem Namen der Skriptdatei.
`$1` bis `$9`	Diese Variablen enthalten die beim Aufruf des Skripts übergebenen Argumente `1` bis `9`.
`$*`	Diese Variable enthält alle beim Aufruf des Skripts übergebenen Argumente als eine einzelne Zeichenkette. Die einzelnen Argumente sind dabei durch Leerzeicheen getrennt.
`$@`	Diese Variable enthält alle beim Aufruf des Skripts übergebenen Argumente als Liste.
`$#`	Diese Variable gibt die Anzahl der beim Aufruf des Skripts übergebenen Argumente an.
`$-`	Diese Variable enthält alle im aktuellen Prozess eingeschalteten Optionsbuchstaben.
`$?`	Diese Variable enthält den *Exit-Status* der zuletzt ausgeführten Anweisung.
`$$`	Diese Variable enthält die Prozess-Nummer der Shell, in der das Skript ausgeführt wird.
`$!`	Diese Variable enthält die Prozess-Nummer des zuletzt erzeugten Hintergrundprozesses.

Die Variable `$$` ist insbesondere für die Erzeugung von temporären Dateien von Bedeutung. Erzeugt ein Skript beispielsweise eine gleichnamige Datei `/tmp/$0` im `/tmp`-Verzeichnis, so würde das Skript bei einem gleichzeitigen Aufruf in verschiedenen Shell-Fenstern die gleiche Datei nutzen und dabei mit großer Wahrscheinlichkeit jeweils Daten der anderen Prozesse überschreiben. Verwendet man hingegen `/tmp/$0.$$` als Namen für die temporäre Datei, so bekommt jede ausführende Instanz des Skripts eine eigene Datei zugewiesen.

Neben den obigen, minimalistisch benannten Variablen existieren weitere vordefinierte Variablen, die häufig in Shell-Skripten eingesetzt werden:

`$EDITOR`	Diese Variable gibt an, welches Programm bevorzugt als Texteditor geöffnet werden soll.
`$HOME`	Diese Variable enthält den Namen des Home-Verzeichnisses des aktuellen Benutzers.
`$PAGER`	Diese Variable gibt an, welches Programm als Pager, also als Anzeigeprogramm für Textdateien geöffnet werden soll
`$PATH`	Diese Variable enthält alle Verzeichnisse, in denen bei Eingabe einer Shell-Anweisung nach einem entsprechenden Programm gesucht wird. Die Namen der einzelnen Verzeichnisse sind durch Doppelpunkte getrennt und werden in der angegebenen Reihenfolge durchsucht.
`$PS1`	In dieser Variablen („Prompt String 1“) wird das Aussehen des Eingabe-Prompts definiert. Üblicherweise steht `$` für normale Benutzer und `#` für den Systemverwalter.
`$PS2`	In dieser Variablen („Prompt String 2“) wird definiert, wie der Eingabe-Prompt im Fall eines Zeilenumbruchs aussehen soll. Üblicherweise wird hierfür das Zeichen `>` verwendet.
`$TERM`	Diese Variable enthält den Namen des aktuellen Shell-Anzeige-Programms.
`$USER`	Diese Variable enthält den Namen des aktuellen Benutzers.

Die aktuellen Werte aller Variablen können mittels der Anweisung `printenv` angezeigt werden; eine einzelne Variable wie `$EDITOR` kann mittels `printenv EDITOR` ausgegeben werden.

Üblicherweise werden die `$EDITOR` und `$PAGER`-Variablen in der Konfigurationsdatei `.bashrc` festgelegt:

```
# Vim als Editor festlegen:
export EDITOR=vim

# Less als Pager festlegen:
export PAGER=less
```

Werden die Variablen nicht vom Benutzer gesetzt, so wird üblicherweise `vi` als Standard-Editor und `cat` als Pager verwendet.

Wer keine Erfahrung mit *Vim* hat, kann an dieser Stelle beispielsweise `pico`, `nano`, `joe` oder `emacs` verwenden, wobei die letzten beiden gegebenenfalls mittels der gleichnamigen Pakete via *apt* installiert werden müssen.

Auswertung von Variablen

In manchen Fällen, beispielsweise beim Arbeiten mit Verzeichnis- und Dateinamen, kann es passieren, dass der Wert einer Variable nahtlos in weiteren Text übergehen soll. In diesem Fall muss der Name der Variablen in geschweifte Klammern gesetzt werden:

```
# Variable definieren:
zwei=ZWEI
```

```
echo eins$zweidrei
# Ergebnis: eins

echo eins${zwei}drei
# Ergebnis: einsZWEIdrei
```

Ist der Variablenname in geschweiften Klammern nicht definiert, so wird sie wie gewöhnlich vom Interpreter ausgelassen (durch „nichts“ ersetzt). Dieses Verhalten des Interpreters kann auf mehrere Arten beeinflusst werden:

- Schreibt man `${variablenname-standardwert}`, so wird an der Stelle der Variablen der angegebene Standardwert eingesetzt, sofern der Variablenname nicht definiert ist; Die Variable bleibt dabei undefiniert.
- Schreibt man `${variablenname=standardwert}`, so wird ebenfalls an der Stelle der Variablen der angegebene Standardwert eingesetzt, sofern der Variablenname nicht definiert ist; die Variable wird dabei allerdings mit dem angegebenen Standardwert neu definiert.
- Schreibt man `${variablenname?fehlermeldung}`, so wird geprüft, ob der angegebene Variablenname definiert ist. Ist er es nicht, so wird das Shellskript abgebrochen und die hinter dem `?` angegebene Fehlermeldung angezeigt. Wird keine Fehlermeldung angegeben, so wird als Standard die Meldung „parameter null or not set“ ausgegeben.

Möchte man im umgekehrten Fall einen bestimmten Wert ausgeben, wenn eine Variable definiert ist, so kann man die Syntax `${variablenname+wert}` verwenden. Beispielsweise Liefert `${one+yes}` den Wert `yes`, wenn die Variable `one` definiert ist, andernfalls wird die angegebene Variable ausgelassen (durch nichts ersetzt).

Quotings

In der Shell haben einfache Anführungszeichen, doppelte Anführungszeichen und so genannte „Backticks“ (`) eine jeweils eigene Bedeutung:

- Innerhalb von doppelten Anführungszeichen kann ein beliebig langer Text als eine einzelne Zeichenkette eingegeben werden. Diese kann sich über mehrere Bildschirmzeilen erstrecken und Leerzeichen beinhalten, ohne dass diesen jeweils ein Backslash vorangestellt werden muss. Die Bedeutung des Dollar-Zeichens bleibt allerdings erhalten, so dass mittels `$variablenname` innerhalb doppelter Anführungszeichen der Wert einer Variablen wie gewohnt ausgewertet werden kann.

  Dateinamen-Erweiterungen, beispielsweise mittels des Stern-Zeichens `*`, sind hingegen innerhalb der Anführungszeichen nicht möglich.
- Innerhalb von einfachen Anführungszeichen kann ebenfalls ein beliebig langer Text als eine einzelne Zeichenkette eingegeben werden. Auch in diesem Fall kann sich der Text über mehrere Bildschirmzeilen erstrecken. Die Besonderheit bei der Benutzung von einfachen Anführungszeichen liegt darin, dass in diesem Fall sämtliche Sonder-

zeichen ihre Bedeutung verlieren, also auch keine Auswertung von Variablen mittels des Dollar-Zeichens `$` erfolgt.

- Backticks werden üblicherweise für Shell-Anweisungen genutzt, wobei die innerhalb der Backticks stehende(n) Anweisung(en) von der Shell durch ihren Rückgabewert ersetzt werden.

Soll das Ergebnis einer Shell-Anweisung wie eine Variable genutzt werden, so kann dies alternativ zur Backticks-Notation auch mittels `$(anweisung)` erfolgen. Diese Schreibweise ist im Allgemeinen sogar vorzuziehen, da sie meist übersichtlicher und somit angenehmer zu lesen ist.

Kontrollstrukturen

Die folgenden Kontrollstrukturen können zur Steuerung eines Shell-Skripts verwendet werden, wenn einzelne Code-Blöcke nur unter bestimmten Bedingungen oder auch mehrfach ausgeführt werden sollen.

Fallunterscheidungen – `if`, `then`, `else`

Mit Hilfe von `if`-Abfragen ist es möglich, Teile eines Shell-Skripts nur unter bestimmten Bedingungen ablaufen zu lassen. Ist die `if`-Bedingung wahr, so wird der anschließende, durch `then` gekennzeichnete Code ausgeführt, bis das Schlüsselwort `fi` (ein umgedrehtes `if`) die bedingte Anweisung abschließt.

Die grundsätzliche Syntax lautet also:

```
if bedingung
then
    anweisungen
fi
```

Optional können nach einem `if`-Block mittels `elif` eine oder mehrere zusätzliche Bedingungen formuliert werden, die jedoch nur dann untersucht werden, wenn die erste `if`-Bedingung falsch ist. Schließlich kann auch eine `else`-Bedingung angegeben werden, die genau dann ausgeführt wird, wenn die vorherige Bedingung (beziehungsweise alle vorherigen Bedingungen bei Verwendung eines `elif`-Blocks) nicht zutreffen.

Insgesamt kann eine Fallunterscheidung beispielsweise folgenden Aufbau haben:

```
if bedingung1
then
    anweisung1

elif bedingung2
then
    anweisung2

else
```

```
    anweisung3

fi
```

Um die Bedingungen zu formulieren, wird häufig die Shell-Anweisung `test` verwendet. Mit dieser lassen sich zum einen Datei-Tests durchführen, zum anderen auch Zahlenwerte und Zeichenketten miteinander vergleichen.

- Um die zu einem Dateinamen gehörende Datei auf eine bestimmte Eigenschaft hin zu überprüfen, lautet die `test`-Syntax wie folgt:

  ```
  test option dateiname
  ```

  Eine Auswahl an häufig verwendeten Prüfoptionen sind in der *Datei-Test-Tabelle* aufgelistet; eine vollständige Liste aller Optionen findet in den `test`-Manpages (`man test`).

`-d`	wahr, wenn Datei existiert und ein Verzeichnis ist
`-e`	wahr, wenn Datei existiert
`-f`	wahr, wenn Datei existiert und regulär ist (kein Verzeichnis, kein Link)
`-h` oder `-L`	wahr, wenn Datei existiert und ein Symlink ist
`-r`	wahr, wenn Datei existiert und lesbar ist
`-s`	wahr, wenn Datei existiert und nicht leer ist
`-w`	wahr, wenn Datei existiert und schreibbar ist
`-x`	wahr, wenn Datei existiert und ausführbar ist

Anstelle von `test option dateiname` kann auch kürzer `[ option dateiname ]` geschrieben werden. In dieser Form kommen Test-Anweisungen sehr häufig bei `if`-Bedingungen vor.

- Um ganzzahlige Werte miteinander zu vergleichen, können die Optionen `-eq`, `-ne`, `-gt`, `-lt`, `-ge`, und `-le` verwendet werden.

  ```
  test zahl1 operator zahl2
  ```

  Die möglichen Vergleichsoperatoren für Zahlen sind in der *Integer-Test-Tabelle* aufgelistet.

`-eq`	wahr, wenn beide Zahlen gleich sind („equal“)
`-ne`	wahr, wenn beide Zahlen nicht gleich sind („not equal“)
`-gt`	wahr, wenn erste Zahl größer als zweite Zahl ist („greater than“)
`-lt`	wahr, wenn erste Zahl kleiner als zweite Zahl ist („less than“)
`-ge`	wahr, wenn erste Zahl größer oder gleich der zweiten Zahl ist („greater or equal“)
`-le`	wahr, wenn erste Zahl kleiner oder gleich der zweiten Zahl ist („less or equal“)

- Um eine einzelne Zeichenkette zu überprüfen, können die Optionen `-z` („zero“) oder `-n` („non-zero“) verwendet werden. Mit `test -z $mystring` wird beispielsweise getestet, ob die in der Variablen `mystring` gespeicherte Zeichenkette die Länge Null hat.

Um zwei Zeichenketten miteinander zu vergleichen, können die Operatoren `==` zum Test auf Gleichheit und `!=` zum Test auf Ungleichheit verwendet werden. Beispielsweise kann mit `if [ $mystring1 == $mystring2 ]` eine Bedingung für die Gleichheit von `mystring1` und `mystring2` formuliert werden.

Möchte man mehrere Teilbedingungen zu einer einzigen Bedingung verknüpfen, können die Optionen `-a` („and“) für eine UND-Bedingung und `-o` („or“) für eine ODER-Bedingung eingesetzt werden. Wird einer (Teil-)Bedingung das Negationszeichen `!` vorangestellt, so wird der Wahrheitswert des Bedingungsterms umgekehrt.

Mehrfach-Unterscheidungen – `case`

Sollen Anweisungen in Abhängigkeit des konkreten Werts einer Variablen oder einer Test-Bedingung ausgeführt werden, so kann das Schlüsselwort `case` genutzt werden. Dieses hat folgende Syntax:

```
case variable in

    muster1) anweisung1 ;;

    muster2) anweisung2 ;;

    muster3) anweisung3 ;;

    *) sonstige-anweisungen ;;

esac
```

Im obigen Beispiel kann anstelle `variable` auch ein Ausdruck stehen, der eine Zeichenkette als Ergebnis liefert.

Trifft ein Muster auf den Wert der Variablen zu, so wird die dahinter angegebene Anweisung ausgeführt. Die `case`-Struktur wird unmittelbar anschließend beendet; bei mehreren passenden Mustern werden somit nur die Anweisungen beim ersten zutreffenden Muster ausgeführt.

Das Muster darf jedes *Suchmuster* beinhalten, das auch für Dateinamen erlaubt ist. Zwei oder mehrere einzelne Teilmuster können dabei mittels `|`-Zeichen zu einem Gesamt-Muster verbunden werden.

Das Suchmuster `*` trifft auf jeden beliebigen Wert zu. Es kann daher verwendet werden, um Anweisungen festzulegen, die genau ausgeführt werden, wenn kein anderer Fall zutrifft. Da nach dem `*`-Muster die `case`-Struktur mit Sicherheit beendet wird, darf es erst am Ende der möglichen Fälle aufgelistet werden. Anschließend wird die `case`-Struktur mittels `esac` (ein umgekehrtes `case`) beendet.

Schleifen – `for`, `while` und `until`

In einer Shell stehen folgende Schleifentypen zur Verfügung:

- Mittels einer `for`-Schleife kann eine Liste von Variablen elementweise abgearbeitet werden. Häufig wird als Liste ein Dateimuster verwendet, beispielsweise würde `for pic in *.png` alle `png`-Dateien des Verzeichnisses in eine Liste speichern und bei jedem Durchlauf der Schleife die jeweils nächste solche Datei in der Variablen `pic` ablegen. Sind alle Elemente der Liste abgearbeitet, wird die `for`-Schleife automatisch beendet.

  Die Anweisungen, die innerhalb der Schleife abgearbeitet werden sollen, werden durch die Schlüsselwörter `do` und `done` begrenzt. Eine `for`-Schleife hat damit insgesamt folgende Form:

```
for varname in var_list
do
    echo "Doing something with $varname ..."
done
```

  In Kurzform, insbesondere bei einer einzelnen Schleifenanweisung, kann eine `for`-Schleife auch in eine Zeile geschrieben werden:

```
for varname in var_list ; do echo "Doing something with $varname ..." ;␣
↪done
```

  Üblicherweise werden `for`-Schleifen zum Durchlaufen einer vorgegebenen Anzahl an Listenelementen verwendet.

- Mittels einer `while`-Schleife kann eine beliebe Anzahl an Anweisungen, solange eine bestimmte Bedingung erfüllt ist, beliebig oft wiederholt werden:

```
while [ $count -le 10 ]
do
    echo "Hallo"
    count=$( expr $count + 1 )
done
```

  Die Bedingung wird vor jedem Schleifendurchlauf geprüft, und sofern diese nicht erfüllt ist, wird die Schleife beendet. Stellt sich die Bedingung schon vor dem ersten Schleifendurchlauf als Falsch heraus, wird die Schleife somit komplett übersprungen.

  Die `expr`-Anweisung wertet dabei den gegebenen arithmetischen Ausdruck aus und gibt das Ergebnis als Rückgabewert zurück. Anstelle `$(expr $count +1)` kann auch kürzer `$(($count + 1))` geschrieben werden. Für komplexere Berechnungen innerhalb eines Shell-Skripts sollte *bc* verwendet werden.

  Eine `while`-Schleife kann beispielsweise verwendet werden, um alle dem Skript beim Aufruf übergebenen Parameter auszulesen:

```
while [ -n $1 ]
do
    echo $1
    shift
done
```

Hierbei bewirkt die Funktion `shift`, dass die Nummerierung der Parameter `$1` bis `$9` nach links verschoben wird, aus `$3` wird beispielsweise `$2` und aus `$2` wird `$1`. Auf diese Weise können auch mehr als 9 übergebene Parameter der Reihe nach abgearbeitet werden.

- Hat man eine Bedingung in der Form `while not`, so kann dafür das Schlüsselwort `until` verwendet werden. Mit `until` wird ebenfalls eine Schleife eingeleitet, wobei die angegebene Bedingung – wie bei einer `while`-Schleife – vor jedem Schleifendurchgang geprüft wird.

```
until [ $count -eq 10 ]
do
    echo "Hallo"
    count=$(expr $count + 1)
done
```

Mit `while` und `until` werden üblicherweise Endlos-Schleifen definiert, die dann zu einem bestimmten Zeitpunkt mittels `break` abgebrochen werden.

break und continue

Um den gewöhnlichen Schleifenverlauf zu verändern, akzeptiert der Shell-Interpreter zwei Schlüsselwörter: `break` und `continue`:

- Mittels `break` wird die Schleife komplett abgebrochen.

  Beispielsweise kann somit eine Endlos-Schleife unterbrochen werden, wenn eine bestimmte Bedingung eintritt:

```
while true
do
  echo "Doing something.."

  if given_condition
  then
      break

done
```

- Mittels `continue` wird der aktuelle Schleifendurchlauf abgebrochen. Die Schleife wird dann mit dem nächsten Schleifendurchlauf fortgesetzt.

  Die `continue`-Anweisung wird häufig in `for`-Schleifen eingesetzt, wenn beispielsweise alle Dateien eines Verzeichnisses abgearbeitet werden und nur in Sonderfällen zur nächsten Datei gegangen werden soll.

Die Anweisung `break` kann bei Bedarf auch mit einer Zahl `n` aufgerufen werden, um in einer verschachtelten Schleife nur die innersten n Ebenen der Schleife zu verlassen (beispielsweise bricht `break 1` nur die innerste Schleifenebene ab). Ebenso kann die Anweisung `continue` mit einer Zahl `n` aufgerufen werden, um insgesamt n Schleifendurchläufe zu überspringen.

Definition von Funktionen

In Shell-Skripten können, ähnlich wie in Programmiersprachen, Kombinationen von mehreren Anweisungen als Funktionen definiert und somit beliebig oft an verschiedenen Stellen eines Skripts aufgerufen werden.

Die grundlegende Syntax zur Definition eigener Funktionen ist folgende:[5]

```
funktionsname ()
{
    anweisungen
}
```

Hat man eine Funktion definiert, so kann sie mittels `funktionsname` aufgerufen werden; die in der Funktion enthaltenen Anweisungen werden dadurch zeilenweise ausgeführt. Wird eine Funktion in der Konfigurationsdatei der Shell definiert, so kann sie von jeder Shell-Sitzung aus wie ein gewöhnliches Programm aufgerufen werden.

Innerhalb eines Shell-Skripts können wiederum Variablen genutzt werden; beispielsweise können so beim Aufruf der Funktion zusätzlich angegebene Argumente (die in den Variablen `$1`, `$2`, usw. gespeichert werden) innerhalb der Funktion genutzt werden.

Persönlich nutze ich beispielsweise die im folgenden vorgestellte Funktion, um aus mit *Inkscape* erstellten Vektor-Graphik–Dateien (SVG) gewöhnliche PNG-Dateien zu erstellen. Die Funktion wird mit dem Dateinamen einer SVG-Datei als Argument aufgerufen, exportiert den kompletten Inhalt als gleichnamige PNG-Datei und komprimiert anschließend die Datei-Größe dieser Datei:

```
# Funktion zum Exportieren einer SVG-Datei:
INK ()
{
    # Export als PNG-Datei mittels inkscape:
    inkscape -z -d 150 -D $1 -e $(basename $1 .svg).png

    # Komprimieren der PNG-Datei mittels pngnq:
    pngnq -n 256 $(basename $1 .svg).png && rm $(basename $1 .svg).png
    mv $(basename $1 .svg)-nq8.png $(basename $1 .svg).png
}
```

Wird die obige Funktion in die `~/.bashrc` beziehungsweise `~/.zshrc` aufgenommen, so kann sie nach einem erneuten Laden dieser Datei mittels beispielsweise `source ~/.zshrc` folgendermaßen genutzt werden:

```
# Funktion INK auf :
INK dateiname.svg
```

Als Ergebnis erhält man in diesem Fall eine zusätzliche Datei `dateiname.png` im selben Verzeichnis.

[5] Funktionsdefinitionen können, ebenso wie Variablen, mit `unset funktionsname` gelöscht werden; sie können allerdings nicht an Unterprozesse exportiert werden.

Beenden einer Funktion mittels `return`

Mit `return num` kann eine Funktion an jeder beliebigen Stelle beendet werden; dabei wird `num` als *Exit-Status* an die Shell zurück geliefert. Gibt es in einer Funktion keine `return`-Anweisung, so entspricht der Exit-Status der letzten Anweisung dem Rückgabewert der Funktion.

Makefiles

Mit dem Programm `make` lassen sich Übersetzungs-Routinen einfach organisieren. Hierzu wird im Ordner des zu übersetzenden Quellcodes eine Datei namens `Makefile` angelegt. Darin werden Anweisungen nach folgender Regel festgelegt:

```
Ziel : Abhängigkeiten
    Anweisung(en) zur Generierung
```

- Die erste Zeile gibt jeweils an, welche (Ziel-)Dateien von welchen (Quell-)Dateien abhängen.
- Nach jeder so festgelegten Abhängigkeitsregel wird in einer oder mehreren Zeilen festgelegt, auf welche Weise die Quelldateien zu übersetzen sind. In jeder Zeile wird üblicherweise eine Anweisung angegeben, mehrere Anweisungen in einer Zeile müssen gegebenenfalls mit einem Strichpunkt getrennt werden.[1] Die Anweisungsliste wird durch eine Leerzeile beendet.

Eine Makedatei enthält üblicherweise mehrere Ziele und zugehörige Übersetzungsregeln.

Der Einsatzbereich von `make` ist wegen dem allgemein gehaltenen Aufbau sehr weitreichend; so findet `make` beispielsweise Einsatz beim Compilieren von eigenem Quellcode, bei Installationsroutinen oder sogar beim Übersetzen von RestructuredText-Dateien in PDF- oder Html-Dateien mittels *Sphinx*.

Beispiel:

- Eine Makefile zum Übersetzen eines C-Programms namens `hallo-welt.c` in ein ausführbares Programm `hallo-welt` sieht im einfachsten Fall folgendermaßen aus:

```
# Makefile-Beispiel 1:

hallo: hallo-welt.c
    gcc -o hallo hallo-welt.c
```

[1] Mehrere Anweisungen werden beispielsweise dann in einzelne Zeile geschrieben, wenn ein Pfadwechsel mittels *cd* stattfindet; dieser bezieht sich nur auf die jeweilige Zeile, bei der nächsten Zeile entspricht der Pfad wieder dem Verzeichnis der Makefile.

Zu lange Zeilen (Faustregel: Über 80 Zeichen) können, wenn sie mit dem Backslash-Zeichen \ abgeschlossen werden, in der nächsten Zeile fortgeführt werden. Dabei sollten die Original-Zeile und die Fortsetzungs-Zeile gleich weit eingerückt werden.

Speichert man das obige Beispiel als `Makefile` im gleichen Ordner, in dem sich auch `hallo-welt.c` befindet, so kann diese aus dem gleichen Ordner heraus in einer Shell mittels `make hallo` compiliert werden.[2]

Mittels des Hash-Zeichens `#` können Makefiles an beliebigen Stellen mit Kommentaren versehen werden, die jeweils bis zum Zeilenende reichen.

Makefiles werden vor allem dann eingesetzt, wenn mehrere Abhängigkeiten existieren oder wenn mehrere Anweisungen zum Übersetzen notwendig sind: Diese müssen bei Verwendung einer Makefile nur einmal eingegeben werden, was den Schreibaufwand meist erheblich reduziert.

Ein weiterer Vorteil von `make` liegt darin, dass nur die Dateien übersetzt werden, die seit dem letzten Übersetzen verändert wurden. Dabei liest `make` nicht den Inhalt der Datei aus, sondern orientiert sich am Änderungsdatum. Dies spart Zeit und bewirkt, dass keine Datei unnötigerweise übersetzt wird. Falls sich jedoch, beispielsweise in einer C-Quellcode-Datei, Abhängigkeiten von anderen Dateien ergeben können, so muss diese C-Datei ebenfalls erneut gespeichert werden, damit die Änderungen wirksam werden.

Makros

Innerhalb eines Makefiles können so genannte Makros definiert werden, die dazu dienen, eine beliebig lange Zeichenkette durch einen Makro-Namen als Kurzbezeichnung zu ersetzen. Üblicherweise werden die Makro-Namen dabei zur besseren Lesbarkeit in Großbuchstaben geschrieben.

Ein Makro kann anschließend an beliebiger Stelle wieder durch den gespeicherten Text ersetzt werden, indem man den Makro-Namen in runde Klammern setzt und das Dollar-Zeichen `$` davor schreibt. So lässt sich das obige Makefile-Beispiel auch folgendermaßen schreiben:

```
# Makefile-Beispiel 2:

DEST = hallo
SOURCE = hallo-welt.c
CC = gcc
CFLAGS = -o

$(DEST): $(SOURCE)
    $(CC) $(CFLAGS) $(DEST) $(SOURCE)
```

Die Verwendung von Makros kann einerseits die Lesbarkeit eines Makefiles erhöhen, andererseits auch die Portierbarkeit auf andere Systeme erleichtern, indem beispielsweise nur der Name des Compilers bei der Makro-Definition ausgetauscht werden muss, aber nicht bei jedem einzelnen Ziel-Anweisungs-Block.

[2] Wird der Inhalt einer Makefile unter einem anderen Dateinamen gespeichert, so muss `make` mit der Option `-f dateiname` aufgerufen werden.

Standard-Makros, Suffix- und Muster-Regeln

Neben eigenen Makros können in Makefiles die vier folgenden vordefinierten Makros verwendet werden:

Makro	Bedeutung
$@	Der Name des Ziels
$?	Alle angegebenen Quelldateien, die jünger als das Ziel sind
$<	Der Name der Quelldatei, welcher die Anweisung ausgelöst hat
$*	Analog, aber ohne Dateiendung

- Die beiden Makros `$@` und `$?` können immer dann eingesetzt werden, wenn wie in den obigen beiden Beispielen ein Ziel explizit angegeben wird.
- Die beiden Makros `$<` und `$*` hingegen werden ausschließlich für so genannte Suffix- oder Muster-Regeln verwendet.

Eine Suffix-Regel kann wiederum immer dann definiert werden, wenn ein bestimmter Dateityp von einem anderen Dateityp abhängt; beispielsweise hängen C-Objekt-Dateien mit der Endung `.o` stets von gleichnamigen C-Quelldateien mit der Endung `.c` ab. Eine Suffix-Regel gibt dann allgemein an, wie man aus einer beliebigen Ausgangsdatei die entsprechende Zieldatei erzeugt. Beispielsweise kann eine einfache Suffix-Regel zur Erzeugung von `.o`-Dateien aus `.c`-Dateien mittels des `gcc`-Compilers folgendermaßen aussehen:

```
.c.o:
    gcc $<
```

Möchte man beispielsweise eine Datei `hallo-welt.c` compilieren, so ist dies bei Verwendung der obigen Makefile mittels `make hallo-welt.o` möglich. Bei diesem Aufruf wird automatisch nach der zur Objekt-Datei `.o` passenden Quelldatei `.c` gesucht, anschließend wird der Compiler mit dem Namen dieser Datei aufgerufen.

Anstelle von Suffix-Regeln können (meist leichter lesbar!) auch Muster-Regeln mittels des `%`-Zeichens definiert werden. Das `%`-Zeichen steht dabei für eine beliebige Zeichenkette. Eine zur obigen Suffix-Regel äquivalente Muster-Regel lautet damit:

```
%.o: %.c
    gcc $<
```

Mittels solcher Regeln lässt sich das Übersetzen von Dateien erheblich vereinfachen, wenn beispielsweise alle Dateien in einem Ordner mit den gleichen Optionen compiliert werden sollen; so muss nicht für jedes Ziel eine extra Regel definiert werden.

Standard-Ziele

In einer Makefile muss nicht jedes Ziel einer Datei entsprechen. Es gibt auch so genannte Standard-Ziele, bei denen bestimmte Aktionen ausgeführt werden sollen, die sich aus dem Namen des Ziels ergeben. Einige solcher Ziele, die häufig in Makefiles auftreten, sind folgende:

Ziel	Bedeutung
`all`	Ziel zum Erzeugen aller anderen Regeln
`clean`	Löschen aller Zwischen- und Zieldateien
`new`	Neuerstellen aller Zieldateien (entspricht `clean` plus `all`)
`usage`	Bedienungs-Text ausgeben

Das Ziel `all` wird zudem standardmäßig von `make` verwendet, wenn kein anderes Ziel angegeben wird. Somit kann jederzeit anstelle von `make all` auch nur `make` eingegeben werden.

Sed und Awk

Unter den vielen Werkzeugen, die in Shell-Skripten eingesetzt werden können, befinden sich auch die zwei Programme `sed` und `awk`, die sich auf sehr vielseitige Art einsetzen lassen daher in diesem Abschnitt gesondert beschrieben haben.

Sed

Sed steht für „Stream Editor“, ist also ein Werkzeug zum Verarbeiten von Datenströmen dar. Im Wesentlichen wird `sed` also verwendet, um mit minimalistischer Syntax exakt auf die jeweilige Aufgabe zugeschnittene Textmanipulationen vorzunehmen. Im folgenden werden einige Einsatzmöglichkeiten der auf Debian, Ubuntu und Linux Mint üblichen Version von `sed` (GNU sed) kurz vorgestellt.

Die allgemeine Sed-Syntax lautet:

```
sed [-optionen] Anweisungen [Dateien]
```

Die einzelnen `sed`-Anweisungen sollten in einfach Anführungszeichen gesetzt werden, um automatische Ersetzungen durch den Shell-Interpreter zu vermeiden. Sollen mehrere Anweisungen ausgeführt werden, so müssen diese mittels einem Strichpunkt (`;`) getrennt werden und `sed` mit der Option `-e` aufgerufen werden.

Wird bei einem `sed`-Aufruf kein Dateiname angegeben, so wird automatisch Text von der Standard-Eingabe eingelesen. Auf diese Weise kann `sed` in Kombination mit einer *Pipe* verwendet werden, um die Ausgabe eines anderen Programms zu bearbeiten:

```
# Lesen von der Standard-Eingabe mittels Pipe:
programmname | sed [-optionen] Anweisungen
```

Üblicherweise gibt `sed` seine Ergebnisse wiederum auf der Standard-Ausgabe aus. Mit der Angabe von `> dateiname` als letztes übergebenes Argument oder auch mittels einer weiteren Pipe und dem Programm *tee* kann die Ausgabe jedoch auch in Textdateien umgelenkt werden:

```
# Lesen von der Standard-Eingabe, Ausgabe in Textdatei:
programmname | sed [-optionen] Anweisungen > dateiname
```

Ebenso können mittels `>> dateiname` als letztes Argument die Ergebnisse eines `sed`-Aufrufs auch an das Ende einer (möglicherweise bereits existierenden) Datei angehängt werden.

Optionen von `sed`

In der folgenden Tabelle sind die wichtigsten Optionen von `sed` aufgelistet:

Option	Bedeutung
`-e anweisung1` `-e anweisung2 ...`	Mehrere Anweisungen nacheinander ausführen. Alternativ dazu kann eine einzelne Anweisung formuliert werden, die mehrere durch Strichpunkte getrennte Teilanweisungen umfasst.
`-f skriptdatei`	Anweisungen aus der angegebenen Skriptdatei anstelle von der Standard-Eingabe lesen
`-i`	Änderungen direkt in angegebene Dateien schreiben („in-place-editing“)
`-n`	Ergebnisse nur dann ausgeben, wenn dies mit der Anweisung `p` explizit verlangt wird.

Wird mittels `-f` eine Skriptdatei (übliche Endung: `.sed`) angegeben, so wird in dieser üblicherweise eine `sed`-Anweisung je Zeile formuliert, eine Trennung einzelner Anweisungen durch Strichpunkte kann in diesem Fall entfallen. Zusätzlich können in eine derartige Skriptdatei Kommentare eingefügt werden, die aus eigenen, mittels des Zeichens `#` eingeleiteten Zeilen bestehen.

Anweisungen von `sed`

Die wohl häufigste Anwendung von `sed` besteht darin, einzelne Wörter oder reguläre Ausdrücke durch andere Begriffe zu ersetzen. Die entsprechende `sed`-Anweisung heißt `s` („substitute“). Ihre Syntax lautet etwa wie folgt:

```
sed 's/alt/neu/g'
```

Die obige Anweisung würde im gesamten an `sed` übergebenen Text nach dem Begriff `alt` suchen und diesen durch `neu` ersetzen. Das Schlüsselzeichen `g` („global“) am Ende der

Anweisung bewirkt, dass nicht nur das erste, sondern alle Vorkommen von `alt` durch `neu` ersetzt werden sollen.[1]

Sollen Ersetzungen in Textstellen vorgenommen werden, die selbst Schrägstriche beinhalten (beispielsweise Pfadangaben), so kann anstelle von `/` auch ein anderes Zeichen als Trennzeichen verwendet werden. Der gleiche Aufruf von Sed sieht beispielsweise mit `#` als Trennzeichen so aus:

```
sed 's#alt#neu#g'
```

Möchte man die Ersetzungen nur in einem bestimmten Bereich, beispielsweise zwischen zwei Zeilennummern, vornehmen, so ist eine Bereichsangabe unmittelbar zu Beginn der `sed`-Anweisung möglich. Diese kann aus einer einzelnen Adresse oder auch zwei Adressen, die einen Bereich markieren, bestehen:

```
# In der Zeile 5 "alt" durch "neu" ersetzen:
5s/alt/neu/g

# In den Zeilen 10-30 "alt" durch "neu" ersetzen:
10,30s/alt/neu/g
```

Bei einer Bereichsangabe kann auch eine der beiden Adressen weggelassen werden, um eine Ersetzung vom Anfang des Textes bis zu einer beziehungsweise ab einer gegebenen Stelle bis zum Ende des Textes zu erreichen:

```
# Ab Zeile 10 "alt" durch "neu" ersetzen:
10,s/alt/neu/g

# Bis Zeile 30 "alt" durch "neu" ersetzen:
,30s/alt/neu/g
```

Bereichsangaben können durch ein angefügtes Ausrufezeichen (`!`) umgekehrt werden. Die Anweisung bezieht sich dann auf alle Zeilen, die außerhalb der Bereichsangabe liegen:

```
# In allen Zeilen außer 10-30 "alt" durch "neu" ersetzen:
10,30!s/alt/neu/g
```

Anstelle von Zeilenangaben können Adressen auch aus Suchmustern bestehen, die ebenfalls zwischen zwei Schrägstrichen angegeben werden:

```
# In allen Zeilen, die "total" enthalten, alt" durch "neu" ersetzen:
/total/s/alt/neu/g

# Zwischen "START" und "END" alle Vorkommnisse von alt" durch "neu" ersetzen:
/START/,/END/s/alt/neu/g
```

[1] Anstelle von `g` kann auch eine beliebige Zahl `n` angegeben werden, um nur das n-te Vorkommen des angegebenen Begriffs zu ersetzen.

Neben dem Schlüsselzeichen `g` gibt es nur noch ein weiteres Schlüsselzeichen für die Substitutions-Anweisung, und zwar `p`. Dieses wird nur in Verbindung mit der Option `-n` verwendet, die eine Ausgabe der `sed`-Ergebnisse grundsätzlich unterdrückt. Das Schlüsselzeichen `p` („print“) am Ende einer `sed`-Anweisung bewirkt, dass das Ergebnis dieser Anweisung dennoch ausgegeben wird.

Auch Bereichsangaben, die aus Suchmustern bestehen, können mittels einem Ausrufezeichen negiert werden. Sowohl in den Bereichsangaben wie ich in den zu ersetzenden Begriffen können zudem *reguläre Ausdrücke* eingesetzt werden.

Weitere gebräuchliche Anweisungen von `sed` sind in der folgenden Tabelle aufgelistet:

Anweisung	Bedeutung
`a`	An die angegebene(n) Stelle(n) den folgenden Text als neue Zeile anfügen („append“). Beispiel: `/adresse/a text`
`c`	Die angegebene(n) Stelle(n) durch den folgenden Text als neue Zeile ersetzen („change“). Beispiel: `/adresse/c text`
`i`	Vor den angegebene(n) Stelle(n) den folgenden Text als neue Zeile einfügen („insert“). Beispiel: `/adresse/i text`
`d`	Die angegebene(n) Stelle(n) löschen („delete“) Beispiel: `/adresse/d`
`p`	Gibt die angegebene(n) Stelle(n) aus („print“); wird üblicherweise in Kombination mit der Option `-n` verwendet.
`q`	Sed innerhalb einer Skriptdatei beenden („quit“).
`r`	Vor den angegebene(n) Stelle(n) den Inhalt der folgenden Datei einfügen („read“). Beispiel: `/adresse/r dateiname`
`y`	An den angegebene(n) Stelle(n) Zeichen aus einer Liste durch andere Zeichen ersetzen („yank“). Beispiel: `/adresse/y/abc/ABC/`
`w`	Schreibt die angegebene(n) Stelle(n) in die folgende Datei („write“). Beispiel: `/adresse/w dateiname`

Weitere Funktionen von `sed` sind in den Manpages beschrieben (`man sed`).

Reguläre Ausdrücke für Sed

In Bereichsangaben, Suchmustern und Ersetzungen können in `sed` so genannte reguläre Ausdrücke eingesetzt werden. Dabei handelt es sich um Kombinationen von normalen Buchstaben und Sonderzeichen, die eine besondere Bedeutung besitzen. Die wichtigsten Sonderzeichen sind in der folgenden Tabelle aufgelistet.

Sonderzeichen	Bedeutung
`^`	Zeilenanfang
`$`	Zeilenende
`.`	ein beliebiges Zeichen (außer dem Newline-Zeichen `\n`)
`[A-Z]`	ein Großbuchstabe
`[a-z]`	ein Kleinbuchstabe
`[0-9]`	eine Ziffer
`[abc123]`	ein Zeichen aus der angegebenen Menge an Buchstaben oder Ziffern
`[^abc123]`	ein beliebiges Zeichen außer der angegebenen Menge an Buchstaben oder Ziffern
`\(` `\)`	Gruppierung der zwischen den Klammern angegebenen Zeichen zu einem einzigen Ausdruck. Die Textstellen, auf welche die einzelnen Gruppierungen zutreffen, können bei Ersetzungen mittels `\1`, `\2`, `\3` usw. wieder eingesetzt werden.
`\{m,n\}`	mindestens m und höchstens n Wiederholungen des vorhergehenden Zeichens oder der vorangehenden Gruppierung. Mit `\{m\}` kann die Anzahl auf genau m, mit `\{m,\}` auf mindestens m festgelegt werden.
`*`	keine, eine oder beliebig viele Wiederholungen des vorhergehenden Zeichens oder der vorangehenden Gruppierung
`\<` `\>`	Wortanfang und Wortende
`&`	Bei Ersetzungen entspricht `&` der gesamten Textstelle, auf welche das angegebene Suchmuster zugetrifft.

In eckigen Klammern kann zur Definition einer Charakter-Klasse auch auch ein anderer Bereich angegeben werden; beispielsweise bezeichnet `[a-m]` einen Kleinbuchstaben zwischen `a` und `m`. Soll ein Zeichen mit Sonderbedeutung, beispielsweise ein Dollar- oder ein Punkt-Zeichen Teil eines regulären Ausdrucks sein, so muss davor ein Backslash-Zeichen gesetzt werden, um die Sonderbedeutung des Zeichens aufzuheben.

Links

- GNU Sed Reference
- Sed-Einzeiler mit Kommentaren
- An Introduction and Tutorial to Sed

Awk

Awk wird bevorzugt verwendet, um tabularische Daten zeilenweise einzulesen und dabei einzelne Zeilen, die bestimmte Muster enthalten, zu bearbeiten. Jedes Awk-Skript, das oftmals nur wenige Zeilen umfasst, besteht also aus Mustern und zugehörigen Aktionen.

Als Suchmuster werden, wie bei Sed, oftmals reguläre Ausdrücke genutzt. Die Formulierung der zugehörigen Aktionen hat einige Ähnlichkeiten mit der Programmiersprache C. Awk nimmt dem Benutzer allerdings viel Arbeit ab: Es liest den Eingabetext automatisch zeilenweise ein und zerlegt jede Zeile anhand eines frei wählbaren Trennzeichens in einzelne Felder (Spalten).

Einfache Awk-Skripte

Einfache Awk-Skripte sind oftmals folgendermaßen aufgebaut:

```
awk [optionen] '/muster/ {print ...}' dateiname
```

Das angegebene Muster kann eine einfache Zeichenkette, aber auch ein regulärer Ausdruck oder eine Bedingung sein. Im obigen Fall würde damit die angegebene Datei zeilenweise eingelesen und einzelne Teile jeder auf das Muster zutreffenden Zeile ausgegeben.

Die einzelnen Felder einer Zeile werden von Awk mit `$1` bis `$9` durchnummeriert, `$0` steht für den gesamten Inhalt einer Zeile. Möchte man beispielsweise von allen Zeilen einer Datei nur die ersten drei Spalte ausgeben, so kann das Muster auch weggelassen werden:

```
# Awk-Print-Anweisung auf alle Zeilen einer Datei anwenden:
awk '{print $1 $2 $3}' dateiname
```

Ebenso kann Awk mittels einer *Pipe* Ausgabedaten eines anderen Programms als Eingabe verwenden. In diesem Fall kann der Dateiname weggelassen werden:

```
# Daten vom Bildschirm anstelle von einer Datei einlesen:
other_programm | awk '/muster/ {anweisungen}'
```

Üblicherweise werden von `awk` so genannte „Whitespace"-Zeichen, also Leerzeichen und Tabulator-Zeichen (`\t`), als Trennzeichen zwischen den einzelnen Feldern einer Zeile interpretiert. Möchte man beispielsweise bei der Verarbeitung einer `.csv`-Datei („Comma Seperated Values") das Zeichen `,` oder `;` als Trennzeichen verwenden, so kann dies mittels der Option `-F` („Field Seperator") angegeben werden:

```
# Das Zeichen ";" als Feldtrennzeichen verwenden:
awk -F ";" '/muster/ {anweisungen}' dateiname
```

In einem Awk-Skript können auch mehrere Muster-Anweisungs-Paare in folgender Form angegeben werden:

```
# Mehrere Muster-Anweisungspaare angeben:
awk [optionen] '/muster1/ {anweisung1} /muster2/ {anweisung2} ...' dateiname
```

In einer Shellskript-Datei können die einzelnen Anweisungen zur besseren Lesbarkeit auch untereinander geschrieben werden:

```
# Mehrere Muster-Anweisungspaare, andere Form:
awk [optionen] '/muster1/ {anweisung1}
```

```
                /muster2/ {anweisung2}
                ... ' dateiname
```

In diesem Fall wird bei jeder eingelesenen Zeile zunächst das erste Muster geprüft und gegebenenfalls der zugehörige Anweisungsblock ausgeführt. Wenn das erste Muster nicht zutrifft, wird geprüft, ob das zweite Muster zutrifft, usw. Sobald ein Muster zutrifft, werden also die entsprechenden Anweisungen ausgeführt, und Awk fährt mit dem Einlesen der nächsten Zeile fort. Die einzelnen Muster-Anweisungs-Paare werden somit als einander ausschließende Entweder-Oder-Abfragen interpretiert. Das vorrangige Suchmuster muss also an erster Stelle stehen, da es sonst gegebenenfalls nicht ausgeführt wird.

Soll ein Anweisungsblock ausgeführt werden, wenn wahlweise das eine und/oder ein anderes Muster auftritt, so können diese in folgender Form angeben werden:

```
# Ausführung, wenn muster1 oder muster2 oder beide zutreffen:
awk [optionen] '/muster1/ || /muster2/ {anweisungen}' dateiname
```

Das Zeichen `||` entspricht somit, ebenso wie in der Programmiersprache C, einem logischen `ODER`. Soll ein Anweisungsblock hingegen nur dann ausgeführt werden, wenn sowohl das eine als auch das andere Muster auftritt, so kann folgende Syntax verwendet werden:

```
# Ausführung nur, wenn sowohl muster1 und muster2 zutreffen:
awk [optionen] '/muster1/ && /muster2/ {anweisungen}' dateiname
```

Das Zeichen `&&` entspricht, ebenso wie in C, einem logischen `UND`. Mittels `||` beziehungsweise `&&` können auch mehr als zwei (Teil-)Muster kombiniert werden; bei Bedarf können runde Klammern gesetzt werden, um die gewünschte Kombination der Ausdrücke zu erreichen (siehe Aussagenlogik).

BEGIN- und END-Anweisungen

In Awk kann man je einen Anweisungsblock einmalig zu Beginn beziehungsweise einmalig am Ende eines Skripts ausführen. `BEGIN`-Anweisungen können beispielsweise dazu genutzt werden, um Header-Zeilen in eine Ausgabe-Datei zu schreiben, bevor die eigentlichen Daten verarbeitet werden.

```
awk 'BEGIN {anweisungen} /muster/ {anweisungen}' dateiname
```

Entsprechend können mittels einer `END`-Anweisung zusätzliche Informationen am Ende der Datenverarbeitung ausgegeben werden:

```
awk '/muster/ {anweisungen} END {anweisungen}' dateiname
```

`END`-Anweisungen sind insbesondere praktisch, wenn mehrere Werte summiert werden und das Ergebnis am Ende ausgegeben werden soll. Dies ist, wie im nächsten Abschnitt beschrieben, durch die Verwendung von Variablen möglich.

Variablen und arithmetische Operationen

In Awk lassen sich auf sehr einfache Weise einzelne Werte in Variablen speichern. Dazu wird folgende Syntax verwendet:

```
varname=wert
```

Die Definition einer Variablen kann an jeder beliebigen Stelle innerhalb eines Awk-Skripts erfolgen. Mittels `print varname` kann der gespeicherte Wert wieder ausgegeben werden. Die Variablen `$0` für den Inhalt der aktuellen Zeile sowie die Variablen `$1` bis `$9` für die einzelnen Felder der aktuellen Zeile sind bereits vordefiniert.

In Awk werden alle Variablen als Zeichenketten interpretiert. Dennoch können einfache arithmetische Operationen auf Variablen angewendet werden; beispielsweise kann mit `awk '{prod=$1*$2 ; print $1 $2 prod}'` eine zweispaltige Datentabelle um eine dritte Spalte ergänzt werden, deren Werte in jeder Zeile dem Produkt der ersten beiden Spalten entspricht.[2]

Einer Variablen kann nicht nur mittels des üblichen Zuweisungsoperators `=`, sondern auch beispielsweise mittels `+=` ein Wert zugewiesen werden. Hierbei wird der bisherige Wert der Variablen um den auf der rechten Seite stehenden Ausdruck erhöht. Da jede neu definierte Variable in Awk zunächst den Wert Null hat, können auf diese Weise beispielsweise alle in einer Spalte stehenden Zahlenwerte aufaddiert werden. Das Ergebnis kann dann mittels eines `END`-Blocks ausgegeben werden:

```
# Dateigrößen des aktuellen Verzeichnisses ausgeben:
# (Die 5. Spalte von `ls -l` gibt die Dateigröße an)
ls -l | awk '{print $5}'

# Alle Werte zur Gesamtgröße aufsummieren:
ls -l | awk '{sum += $5} END {print "Gesamt:\t" sum}'
```

Reguläre Ausdrücke für Awk

In den angegebenen Mustern können auch in Awk reguläre Ausdrücke eingesetzt werden; damit sind Kombinationen von normalen Buchstaben und Sonderzeichen gemeint, wobei letztere die eine besondere Bedeutung besitzen. Die wichtigsten Sonderzeichen sind in der folgenden Tabelle aufgelistet.

[2] Möchte man die einzelnen Spalten bei der Ausgabe durch Tabulator-Zeichen `"\t"` getrennt haben, so ist dies mittels `awk '{prod=$1*$2 ; print $1 "\t" $2 "\t" prod}'` möglich.

Sonderzeichen	Bedeutung
`^`	Zeilenanfang
`$`	Zeilenende
`.`	ein beliebiges Zeichen
`[A-Z]`	ein Großbuchstabe
`[a-z]`	ein Kleinbuchstabe
`[0-9]`	eine Ziffer
`[abc123]`	ein Zeichen aus der angegebenen Menge an Buchstaben oder Ziffern.
`[^abc123]`	ein beliebiges Zeichen außer der angegebenen Menge an Buchstaben oder Ziffern
`( )`	Gruppierung der zwischen den Klammern angegebenen Zeichen zu einem einzigen Ausdruck.
`|`	entweder der vor unmittelbar vor oder unmittelbar nach `|` stehende Audruck (oder die entsprechende Gruppierung)
`+`	eine oder beliebig viele Wiederholungen des vorhergehenden Zeichens oder der vorangehenden Gruppierung
`*`	keine, eine oder beliebig viele Wiederholungen des vorhergehenden Zeichens oder der vorangehenden Gruppierung
`?`	kein oder genau ein Vorkommen des vorhergehenden Zeichens oder der vorangehenden Gruppierung
`{m, n}`	mindestens m und höchstens n Wiederholungen des vorhergehenden Zeichens oder der vorangehenden Gruppierung. Mit `{m}` kann die Anzahl auf genau m, mit `{m,}` auf mindestens m festgelegt werden.

Im Gegensatz zu den *regulären Ausdrücken für Sed* haben runde und geschweifte Klammern standardmäßig die oben angegebene Sonderbedeutung; soll das jeweilige Zeichen an sich Teil eines regulären Ausdrucks sein, so muss davor ein Backslash-Zeichen gesetzt werden.

Bedingungen als Muster

Nicht nur reguläre Ausdrücke, sondern auch Bedingungen können als Muster zur Auswahl der zu bearbeitenden Zeilen genutzt werden. Sollen beispielsweise alle Zeilen einer Tabelle ausgegeben werden, deren Wert in der dritten Spalte ≥ 50 ist, so könnte man folgendes schreiben:

```
# Print-Anweisung unter einer bestimmten Bedingung ausführen:
awk '$3 >= 50 {print $0}' dateiname
```

Für Werte-Vergleiche können folgende Operatoren genutzt werden:

Operator	Beschreibung
==	Test auf Wertgleichheit
!=	Test auf Ungleichheit
<	Test, ob kleiner
<=	Test, ob kleiner oder gleich
=>	Test, ob größer oder gleich
>	Test, ob größer

Auch bei Werte-Vergleichen können mehrere Bedingungen mittels `&&` als `UND`-Verknüpfung beziehungsweise `||` als `ODER`-Verknüpfung zu einer Gesamt-Bedingung kombiniert werden; ebenso sind Kombinationen von Werte-Vergleichen und normalen Suchmustern oder regulären Ausdrücken möglich. Zur Gruppierung einzelner Teilbedingungen können wiederum runde Klammern gesetzt werden.

Der Istgleich-Operator `==` kann zudem verwendet werden, um eine Spalte mit einer Zeichenkette zu vergleichen, beispielsweise `$1 == "Hallo"`.

Links

- Awk Tutorial (PDF)
- Einführung in Awk 1
- Einführung in Awk 2 (PDF)
- Awk Wikibook

Die Z-Shell

. . . to be continued . . .

Links

- Z-Shell Reference Card (en.)
- ZSH-Lovers – Tips, Tricks and Examples for the Z-Shell (en.)

Hilfreiche Shell-Anwendungen

Der Terminal-Multiplexer `tmux`

`tmux` erlaubt es (ebenso wie sein Vorgänger *screen*), aus einem einzelnen Shell-Fenster eine ganze Shell-Session mit mehreren Fenstern und Unterfenstern zu erstellen. Ein wesentliche Weiterentwicklung gegenüber `screen` ist die verbesserte Konfigurierbarkeit.

Mittels *aptitude* kann `tmux` über das gleichnamige Paket installiert werden:

Installation und Einführung

Tmux kann via *aptitude* über das gleichnamige Paket `tmux` installiert werden:

```
sudo aptitude install tmux
```

Gibt man anschließend in einem Shell-Fenster `tmux` ein, so wird eine neue `tmux`-Session gestartet. Am unteren Rand des Shell-Fensters wird dabei eine (standardmäßig) grüne Informationsleiste eingeblendet, die als eine Art „Tableiste“ angesehen werden kann. Zunächst ist nur ein Fenster geöffnet, wobei die Nummerierung in `tmux` (standardmäßig) mit Null beginnt.

Um ein weiteres Fenster zu erstellen, kann entweder `tmux new-window` eingegeben werden oder – bei den Standard-Einstellungen – `Ctrl b` und anschließend `c` („create“) gedrückt werden. In der Infoleiste werden nun zwei Fenster aufgelistet, zwischen denen mit `Ctrl b` und `n` beziehungsweise `p` („next“ beziehungsweise „previous“) gewechselt werden kann.

Konfiguration von tmux

Standardmäßig leitet `Ctrl b` eine `tmux`-Anweisung ein; es kann allerdings auch jede andere Tastenkombination hierfür verwendet werden. Dazu wird eine Datei `~/.tmux.conf` angelegt:

```
touch ~/.tmux.conf
```

In diese Datei kann beispielsweise folgender Eintrag aufgenommen werden, um den `tmux`-Hotkey auf `Ctrl d` festzulegen:

```
# Neu-Definition der Präfix-Taste:
set-option -g prefix C-d
```

Damit dieser Eintrag wirksam wird, muss die Konfigurationsdatei neu geladen werden. Dies erfolgt mittels `tmux source`:

```
tmux source ~/.tmux.conf
```

Um einfacher mit der Konfigurationsdatei `.tmux.conf` zu experimentieren und die dortigen Einträge unmittelbar „live“ testen zu können, empfiehlt es sich, dort eine Tastenkombination für das Neuladen der Konfigurationen zu definieren:

```
# Tastenkürzel zum Laden der Konfigurationsdatei:
bind-key r source-file ~/.tmux.conf \; display-message "tmux.conf reloaded."
```

Wird die Konfigurationsdatei noch einmal mit `tmux source ~/.tmux.conf` neu geladen, so kann man künftig dafür auch einfacher `Ctrl d` und `r` eingeben.

Tastenkürzel ohne Präfix

Mit `bind-key` lassen sich auch Tastenkürzel definieren, die ohne den tmux-Hotkey auskommen. Dies kann beispielsweise verwendet werden, um mittels `Shift Links` und `Shift rechts` zwischen den offenen Tabs zu wechseln, mit `Shift unten` einen neuen Tab zu erzeugen und mit `Shift oben` dem aktuellen Tab manuell einen festen Namen zu geben. Die entsprechenden Einträge in der Konfigurationsdatei lauten:

```
# Schnelles Öffnen und Umbenennen von Fenstern:
bind-key -n S-Down new-window -c "#{pane_current_path}"
bind-key -n S-Up command-prompt -I "rename-window "

# Schnelles Navigieren zwischen Fenstern:
bind-key -n S-Left previous-window
bind-key -n S-Right next-window
```

Die Option `-n` bedeutet dabei, dass das Tastenkürzel nicht den tmux-Hotkey als Präfix erwartet. In gleicher Weise kann definiert werden, dass mittels `Ctrl Shift Links` beziehungsweise `Ctrl Shift Rechts` das aktuelle Fenster in der Infoleiste nach links beziehungsweise rechts verschoben wird:

```
# Schnelles Verschieben von Fenstern
bind-key -n C-S-Left swap-window -t -1
bind-key -n C-S-Right swap-window -t +1
```

Soll ein Fenster geschlossen werden, kann entweder `Ctrl d x` oder `exit` eingegeben werden. Entstehen durch das Schließen von Fenstern „Lücken“ in der Nummerierung der Fenster, so kann für eine automatische Re-Nummerierung `tmux move-window -r` aufgerufen werden oder in der Konfigurationsdatei ein entsprechendes Tastenkürzel definiert werden, beispielsweise `bind-key m move-window -r`.

Weitere Optionen

Folgende Optionen für die Konfigurationsdatei können ebenfalls nützlich sein:

```
# Anzahl an History-Einträgen auf 10000 erhöhen:
set-option -g history-limit 10000

# Nummerierung der Fenster und Teilfenster jeweils mit 1 beginnen:
set-option -g base-index 1
set-window-option -g pane-base-index 1

# Pfeiltasten sofort nach Fenster-Wechsel freigeben:
set-option -g repeat-time 0

# Maus-Unterstützung aktivieren:
set-window-option -g mode-mouse on
set-option -g mouse-select-window on
set-option -g mouse-select-pane on
set-option -g mouse-resize-pane on

# Farb-Optionen für Shell-Fenster:
set-option -g default-terminal screen-256color

# Inhalt der Infoleiste ändern:
set -g status-interval 2
set -g status-left '[#S]'
set -g status-right '%l:%M'
set -g status-utf8 on
set-option -g status-justify left
set-window-option -g window-status-current-format '#I:#W#F'
set-window-option -g window-status-format '#I:#W#F'

# Aussehen der Infoleiste ändern:
set-option -g status on
set-option -g status-bg blue
set-option -g status-fg white
set-window-option -g window-status-current-bg magenta

# Aussehen der Kommandozeile ändern:
set -g message-fg white
set -g message-bg black
set -g message-attr bright

# Aussehen von Teilfenstern ("Panes") anpassen:
set -g pane-border-fg green
set -g pane-border-bg black
set -g pane-active-border-fg green
set -g pane-active-border-bg black

# Aktive Shell-Fenster visuell hervorheben:
setw -g monitor-activity on
```

```
set -g visual-activity on

# Automatische Neu-Nummerierung der Fenster aktivieren:
# (Beispielsweise nach dem Schließen eines Fensters)
set -g renumber-windows on
```

Bei Verwendung der `zsh` sollten zudem die folgenden Beiträge in der Datei `.tmux.conf` stehen, um ein automatisches Umbenennen von Fenstern bei einem Programmaufruf oder einem Verzeichniswechsel zu verhindern:

```
set-window-option -g automatic-rename off
set-option -g allow-rename off
```

Mit den obigen Einstellungen können zum einen Fenster in der Infoleiste auch mit der Maus ausgewählt werden, zum anderen wird das Aussehen der Infoleiste angepasst: Die Hintergrundfarbe wird auf allgemein auf blau, die Farbe des aktuellen Fensters auf magenta festgelegt; andere aktive Fenster werden in weißer Farbe markiert. Die Nummerierung der Fenster beginnt von nun an mit `1`, rechts wird die aktuelle Uhrzeit angezeigt.

Die Verwendung von Teilfenstern wird im Abschnitt *Teilfenster („Panes“)* näher beschrieben.

Tab-Vervollständigung

Bei Verwendung der `bash`-Shell ist für `tmux` leider keine Tab-Vervollständigung der einzelnen möglichen Anweisungen vordefiniert. Beispielsweise kann in einer Shell nicht `tmux bi<TAB>` eingegeben werden, um eine Vervollständigung zu `tmux bind-key` zu erhalten. Dieses nützliche Feature kann man für die `bash`-Shell allerdings durch das folgende Skript erhalten:[1]

`tmux-completion.sh`

Speichert man diese Datei beispielsweise im Verzeichnis `~/bin`, so sollte man folgenden Eintrag in der Datei `~/.bashrc` hinzufügen, damit die Vervollständigung automatisch geladen wird:

```
# Tmux-Completion laden:
source ~/bin/tmux-completion.sh
```

Anschließend können in einem neuen Shell-Fenster tmux-Anweisungen automatisch mit `Tab` ergänzt oder, falls keine eindeutige Ergänzung möglich ist, durch Drücken von `Tab Tab` alle möglichen Ergänzungen eingeblendet werden.

[1] Bei Verwendung der `zsh` mit der `oh-my-zsh`-Konfiguration funktioniert die `tmux`-Vervollständigung automatisch.

Sitzungen, Fenster und Teilfenster

Am linken Rand der Infoleiste wird bei Verwendung der im vorherigen Abschnitt beschriebenen Konfiguration der Name der tmux-Session angezeigt. Wird `tmux` ohne weitere Argumente aufgerufen, so werden die einzelnen Sessions automatisch durchnummeriert. Wird allerdings `tmux new -s myname` aufgerufen, so wird eine neue Session mit der Bezeichnung `myname` erzeugt. Dies ist insbesondere nützlich, wenn ein tmux-Fenster beispielsweise für das Vim-Vicle-Plugin als „Code-Empfänger“ verwendet werden soll.

Auch aus einem anderen Grund sollten tmux-Sessions benannt werden: Wird beispielsweise das Shell-Hauptfenster geschlossen, das die tmux-Session beinhaltet, so ist die tmux-Session nicht verloren, sondern hat lediglich den status „deattached“. Gibt man in einem anderen Shell-Fenster `tmux list-sessions` ein, so werden alle Sessions aufgelistet. Hieß die vermeintlich geschlossene tmux-Session beispielsweise `myname` genannt, so kann sie folgendermaßen wieder reaktiviert („attached“) werden:

```
tmux attach-session -t myname
```

Die Option `-t` gibt hierbei die Zielsession („target“) an. Die Session wird so mitsamt allen Fenstern und Teilfenstern wieder geladen. Auf diese Weise kann zum Beispiel ein Shell-Fenster weiter aktiv bleiben, auch wenn der Benutzer abgemeldet ist.

Fenster und Teilfenster („Panes“)

Jedes tmux-Fenster kann bei Bedarf in zwei oder mehrere Unterfenster („Panes“) aufgeteilt weden. Dazu sind folgende Tastenkürzel-Definitionen in der Konfigurationsdatei `~/.tmux.conf` nützlich:

```
bind - split-window -v # Horizontales Halbieren des aktuellen Fensters
bind | split-window -h # Vertikales Halbieren des aktuellen Fensters
```

Templates für neue Tmux-Sitzungen

Tmux-Sitzungen können leider nicht gespeichert werden, wenn der Computer ausgeschaltet wird. Es gibt allerdings mit Tmuxinator ein ergänzendes Programm, mit dem auf einfache Weise Templates für neue `tmux`-Sitzungen erstellt werden können.

Zur Installation von `tmuxinator` gibt man in einem Shell-Fenster folgende Zeile ein:

```
sudo gem install tmuxinator
```

Um ein neues Template für beispielsweise eine Sitzung mit Namen `work` zu erzeugen, gibt man folgendes ein:

```
tmuxinator new work
```

Hierdurch wird im Ordner `~/.tmuxinator` eine Datei `top.yml` angelegt, die eine Beispiel-Struktur sowie in auskommentierter Form mögliche Optionen mitsamt Beschreibung ent-

hält. Beispielsweise kann so festgelegt werden, welche Fenster in der Sitzung vorkommen sollen, welche Verzeichnisse in den einzelnen Fenstern aktiv und welche Programme ausgeführt werden sollen:

```
# ~/.tmuxinator/work.yml

name: work
root: ~/data/homepage/work

windows:
  - linux: vim
    root: ~/data/homepage/work/linux
  - python:
    root: ~/data/homepage/work/informatik/python
  - mc: mc
```

Eine gemäß diesem Beispiel definierte neue `tmux`-Sitzung umfasst drei Fenster, die `linux`, `python` und `mc` genannt sind. Im ersten Fenster soll dabei `vim` aufgerufen werden, im dritten `mc`; die aktiven Arbeitsverzeichnisse können entweder explizit angegeben werden oder entsprechen dem angegebenen `root`-Pfad.

Die Sitzung kann folgendermaßen erzeugt:

```
tmuxinator work

# oder kürzer:

mux work
```

Da beim Start der Sitzungen beispielsweise auch Dienste gestartet und innerhalb der Fenster optional weitere Unterfenster erzeugt werden können, kann `tmuxinator` ein Speichern der `tmux`-Sitzungen zumindest weitgehend ersetzen.

Das Dokumentations-System `sphinx`

Sphinx ist ein in Python geschriebenes Shell-Programm, das aus einer bzw. mehreren Textdateien mit RestructuredText-Syntax auf dem lokalen Rechner wahlweise eine PDF-Datei oder eine beziehungsweise mehrere HTML-Dateien erzeugen kann.[1]

Da die RestructuredText-Syntax minimal und leicht erlernbar ist, ermöglicht es Sphinx, Dokumente mit nicht einmal dem halben gewöhnlichen Aufwand in einer Online-Version für Webbrowser sowie in einer (druckbaren) PDF-Version zu publizieren.

[1] Um die erzeugten HTML-Dateien im Internet zu publizieren, müssen sie lediglich in einen gemeinsamen Ordner auf einem Webserver kopiert werden. Weist man diesem Ordner anschließend über einen Domain-Anbieter eine feste Webadresse (URL) zu, so ist die Seite bereits fertig!

Dursuchen von RST-Dateien

Die Quelldateien eines Sphinx-Projekts sind reine Textdateien mit der Endung `.rst` (für RestructuredText); diese lassen sich schnell und einfach nach Inhalten durchsuchen. Persönlich habe ich mir dazu folgende Abkürzung in der Konfigurationsdatei `~/.zshrc` definiert:

```
alias rstgrep='find ./ -name "*.rst" | xargs grep'
```

In einer Shell kann damit mittels `rstgrep Suchbegriff` nach einem Begriff oder einem regulären Ausdruck in allen `rst`-Dateien eines Projekts (inklusive aller Unterverzeichnisse) gesucht werden; als Ergebnis bekommt man jede Zeile einer `rst`-Datei angezeigt, die den Suchbegriff enthält. Durch die Verwendung von *grep* können auch reguläre Ausdrücke genutzt werden; beispielsweise würde `rstgrep "^Hallo *"` jede Zeile ausgeben, die mit „Hallo“ beginnt. Zusätzliche `grep`-Optionen können ebenfalls wie üblich genutzt werden, würde *rstgrep -li Suchbegriff‘* („list“, „ignore-case“) anstelle der zutreffenden Zeilen nur die Namen der Dateien auflisten, die den Suchbegriff ohne Berücksichtigung der Groß-/Kleinschreibung enthalten.

Installation von Sphinx

Sphinx sowie einige nützliche Zusatz-Pakete können unter Linux folgendermaßen installiert werden:

```
sudo aptitude install python3-setuptools python3-numpy \
    python3-matplotlib python3-docutils dvipng latexmk

sudo pip3 Sphinx
```

Mit `sudo pip3 -U Sphinx` („Update“) kann Sphinx jederzeit auf den aktuellsten Stand gebracht werden.

Zur Erzeugung von PDF-Druckversionen genügt bereits ein minimales LaTeX-System, wie es nach Installation der oben genannten Pakete automatisch vorhanden ist. Um beispielsweise in naturwissenschaftlichen Publikationen einen umfangreichen mathematischen Formelsatz nutzen zu können, sollte bei Bedarf ein volles LaTeX-System installiert werden.

Sphinx-Quickstart

Um ein neues Sphinx-Projekt zu starten, kann man entweder einen bestehenden Projekt-Ordner kopieren und modifizieren, oder ein neues Verzeichnis anlegen und in diesem mittels der Shell-Anweisung `sphinx-quickstart` ein neues Grundgerüst erstellen:

```
mkdir projektordner && cd projektordner

sphinx-quickstart
```

Sphinx erstellt daraufhin einige notwendige Hilfsdateien, beispielsweise eine *Makefile* und eine Konfigurationsdatei namens `conf.py`, in der auch zu einem späteren Zeitpunkt Einstellungen für das jeweilige Projekt festgelegt werden können.

Beim Aufruf von `sphinx-quickstart` erscheinen zunächst einige Benutzer-Abfragen, mit denen der Name des Projekts sowie andere grundlegende Einstellungen vorgenommen werden können:

- Bei der ersten Abfrage soll der Nutzer angeben, in welchem Pfad das Projekt initiiert werden soll. Üblicherweise startet man `sphinx-quickstart` bereits im Projekt-Verzeichnis, so dass kein expliziter Pfad angegeben werden muss, sondern die Vorgabe `[.]` durch Drücken der `Enter`-Taste bestätigt werden kann:

```
Welcome to the Sphinx 1.6.2 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Enter the root path for documentation.
> Root path for the documentation [.]:
```

- Bei den beiden nächsten Abfragen geht es darum, ob für die Quelldateien ein eigener Ordner angelegt werden soll und wie besondere Ordner, die beispielsweise Logos oder Templates beinhalten, gekennzeichnet werden sollen. Bei beiden Fragen sind meiner Meinung nach die Standard-Vorgaben empfehlenswert, es genügt somit jeweils eine Bestätigung mit der `Enter`-Taste.

  Die Quelltexte der Webseite werden bei diesen Standard-Einstellungen im Projektverzeichnis beziehungsweise in Unterverzeichnissen abgelegt; der von Sphinx erzeugte HTML- beziehungsweise LaTeX-Code wird hingegen in die separaten Unterverzeichnisse `_build/html` beziehungsweise `_build/latex` geschrieben.

- Bei den nächsten Abfragen muss der Projekt- sowie der Autorname der Dokumentation angegeben werden; die Angabe einer Versionsnummer ist optional und kann auch zu einem späteren Zeitpunkt in der Konfigurationsdatei `conf.py` vorgenommen werden.

  Als Sprache kann wahlweise `en` für Englisch, `de` für Deutsch, oder ein anderes Sprachbeziehungsweise Länderkürzel gesetzt werden (eine Übersicht über die unterstützten Sprachen gibt es hier).

- Bei den beiden nächsten Abfragen wird eine Datei-Endung für die Quelltext-Dateien sowie der Name der grundlegenden Index-Datei festgelegt. Auch hier ist es empfehlenswert die jeweiligen Vorgaben mit der `Enter`-Taste zu bestätigen.

  Die Datei `index.rst` im Projekt-Verzeichnis beinhaltet im Wesentlichen ein Inhaltsverzeichnis („toctree“), welcher die Einhänge-Punkte der übrigen Quelltext-Dateien festlegt.

- Bei der nächsten Abfrage wird festgelegt, ob ein Epub-Builder gewünscht ist oder nicht. Gewöhnlich kann hier die Vorgabe `[n]` mit der `Enter`-Taste bestätigt werden.

- Bei der nächsten Abfrage kann man festlegen, welche Sphinx-Module für das Dokumentations-Projekt genutzt werden sollen; die Auswahl kann ebenfalls später in der Konfigurationsdatei `conf.py` überarbeitet werden.

  Bei mir persönlich würde die Modul-Abfrage folgendermaßen ausfallen:

  > **autodoc:** `y` Dieses Modul ist für die Dokumentation von Python-Programmen gedacht – hierbei kann eine Dokumentation automatisch anhand der Docstrings der jeweiligen Funktionen beziehungsweise Module erstellt werden.

  > **doctest:** `n` Python ermöglicht es, in die Docstrings von Funktionen Tests einzubauen. Ich persönlich verwende lieber Unittests und nutze dieses Feature daher nicht.

  > **intersphinx:** `y`

    Intersphinx-Mappings ermöglichen es, von einer Sphinx-Dokumentation aus auf andere Sphinx-Dokumentationen zu verweisen.

    Hierbei wird in der Konfigurationsdatei `conf.py` unter dem Begriff `intersphinx_mapping` festgelegt, welche externen Projekte mit welchem Kürzel genutzt werden sollen. Ein solcher Eintrag könnte beispielsweise wie folgt aussehen:

    ```
    intersphinx_mapping = {
        'sphinx': ('http://www.sphinx-doc.org/en/stable', None),
        'gwm': ('http://grund-wissen.de/mathematik', None),
        'gwp': ('http://grund-wissen.de/physik', None),
    }
    ```

    Innerhalb der Dokumentation kann dann beispielsweise mittels `:ref:`Mechanik <gwp:Mechanik>` auf den Mechanik-Teil der Physik-Dokumentation im Grund-Wissen-Projekt verwiesen werden.[1]

  > **todo:** `n` Dieses Modul ermöglicht es, Todo-Notizen in die Dokumentation aufzunehmen eine Übersichtsliste daraus zu erstellen. Persönlich vermerke ich mir Todos allerdings lieber als Kommentare, die dann im fertigen Dokument nicht erscheinen.

  > **coverage:** `n` Dieses Modul ist ebenfalls für die Dokumentation von Python-Programmen gedacht, und ermöglicht eine Anzeige, wie viele der definierten Funktionen bereits über eine Dokumentation (einen Docstring) verfügen.

[1] Die Angaben können zu jedem späteren Zeitpunkt in der Konfigurationsdatei `conf.py` geändert werden.

Durch die Vergabe von Versionsnummern kann beispielsweise bei der Dokumentation von Software-Quellcode sichergestellt werden, dass eine Anleitung auch zur jeweiligen Software-Version passt. Auch bei allgemeinen Dokumentationsprojekten ist eine Versionsnummer sinnvoll, um den jeweiligen Entwicklungsstand aufzuzeigen; mit einem Versions-Upgrade können außerdem eine Rundmail über einen Verteiler, ein neuer Commit eines Versionsverwaltungs-Programms, ein Weblog-Eintrag o.ä. verbunden werden.

> **imgmath:** `y` Bei Verwendung dieses Moduls werden mathematische Formeln für die HTML-Version als `png`-Dateien gerendert; diese werden dann an den jeweiligen Stellen automatisch eingefügt.

  Diese Option hat als Nachteil, dass bei technischen Dokumentationen unter Umständen viele (sogar tausende) `png`-Dateien erstellt werden, was ein Hochladen des Projekts auf den Webserver verlangsamt. Der Vorteil ist hingegen, dass die Anzeige im Webbrowser auch ohne zusätzliche Javascript einwandfrei funktioniert.

> **mathjax:** `n` Bei Verwendung dieses Moduls werden mathematische Formeln in der HTML-Version so ausgegeben, dass sie erst im Browser des Lesers mittels Javascript gerendert werden.

  Je nach Vorlieben sollte man sich *entweder* für die `imgmath`- oder für die `mathjax`-Option entscheiden.

> **ifconfig:** `n` Dieses Modul ermöglicht es, bestimmte Inhalte nur dann in der Dokumentation zu berücksichtigen, wenn entsprechende Konfigurationen in der Datei `conf.py` vorliegen. Persönlich habe ich dieses Feature bislang nicht benötigt.

> **viewcode:** `y` Dieses Modul ermöglicht es, die Dokumentation von (Python-)Quellcode mit den jeweiligen Stellen des Quellcodes selbst zu verknüpfen; dies ist für die Dokumentation von Open-Source-Programmen durchaus nützlich.

- Bei der letzten Abfrage gibt man an, ob eine *Makefile* (für Linux-Systeme) oder eine `Commandfile` (für Windows-Systeme) angelegt werden soll; diese Entscheidung muss je nach eingesetztem Betriebsystem individuell getroffen werden.

Anschließend wird das Projekt von Sphinx fertig angelegt und kann beliebig gestaltet beziehungsweise mit Inhalten gefüllt werden.

Aufruf von Sphinx

Ein bestehendes Projekt (beispielsweise ein selbst erstelltes oder ein von *Github* geclontes) kann auf einfache Weise als Webseite oder PDF-Datei ausgegeben werden. Hierzu wechselt man in einer Shell in das Projekt-Verzeichnis und gibt folgendes ein:

```
# HTML-Dateien erzeugen:
make html

# LaTeX-Code erzeugen:
make latex

# LaTeX-Code erzeugen und daraus eine PDF-Datei erstellen:
make latexpdf
```

Treten aufgrund einer fehlerhaften RST-Syntax während des Übersetzens Fehler auf, so werden diese mit einer kurzen Erläuterung und der Angabe der den Fehler verursachenden Stelle auf dem Bildschirm ausgegeben.

Die neu erstellten Dateien werden vons `sphinx` bei Verwendung der oben genannten Konfiguration im `_build`-Verzeichnis innerhalb des Projektpfads abgelegt. Je nach Ausgabe-Variante können die erstellten folgendermaßen aufgerufen werden:

```
# Erstellte HTML-Seiten mit Webbroswer "firefox" öffnen:
firefox _build/html/index.html

# Erstellte PDF-Datei mit PDF-Betrachter "evince" öffnen:
evince _build/latex/projekttitel.pdf
```

Der Name des PDF-Dokuments wird in der Konfigurationsdatei `conf.py` unter der Rubrik `latex_documents` festgelegt.

Um den bestehenden Build eines Projekts zu entfernen, beispielsweise nach einem Umbenennen mehrerer Quelldateien oder einer neuen Ordnerstruktur, kann Folgendes eingegeben werden:

```
make clean
```

Anschließend können mittels `make html`, `make latex` oder `make latexpdf` neue Builds erstellt werden.

Projekt auf nicht funktionierende Links prüfen

Auf folgende Weise kann ein bestehendes Projekt hinsichtlich nicht funktionierender Web-Links überprüft werden:

```
make linkcheck
```

Dieser Aufruf gibt auf dem Bildschirm alle Links zu nicht mehr existierenden oder permanent umgeleiteten Seiten aus. Dieses Feature sollte in regelmäßigen Abständen genutzt werden, um den Besuchern der Seite unnötige `404: Seite nicht gefunden`-Fehlermeldungen zu ersparen; auch Suchmaschinen werten einen möglichst hohen Anteil an funktionierenden Links als Kriterium für die Aktualität einer Seite.

Intersphinx-Mappings aktualisieren

Bei der Verwendung von Sphinx ist es möglich, Links auf Begriffe aus anderen Sphinx-Dokumentationen zu setzen; dies wird als `Intersphinx-Mapping` bezeichnet.

Mit den normalen Einstellungen werden die Index-Kataloge der angegebenen Projekte nur beim erstmaligen Erstellen eines Projekts geladen. Kommen bei den externen Projekten weitere Begriffe hinzu, so kann also nur dann mittels eines Intersphinx-Mappings darauf verwiesen werden, wenn explizit geprüft wird, ob sich Änderungen in den angegebenen Projekten ergeben haben. Dies kann durch folgende Änderung in der `Makefile` des Projekts erreicht werden:

```
# Original
# SPHINXOPTS =
```

```
# Intersphinx-Seiten auf Änderungen prüfen:
SPHINXOPTS = -E
```

Durch die ergänzende Angabe der Option `-E` werden beim Aufruf von `make html`, `make latex` oder `make latexpdf` alle externen Index-Kataloge neu eingelesen. Dies kann den Übersetzungs-Prozess erheblich verlangsamen und sollte daher nur bei Bedarf kurzzeitig geändert werden.

Einzelne Dateien mit `rst2latex` konvertieren

Hat man unter Linux das Paket `python3-docutils` installiert, so stehen neben Sphinx auch die Konverter `rst2html` und `rst2latex` zur Verfügung, die jeweils eine einzelne Quelldatei in ein HTML- beziehungsweise LaTeX-Dokument übersetzen.

Für die Verwendung von `rst2latex` habe ich mir eine *Makefile* mit folgendem Inhalt gebastelt:

```
# Datei: rstmakefile

%: %.rst
    rst2latex $< > $*.tex
    pdflatex $*.tex
    rm $*.aux $*.log $*.out
```

In einer Shell kann man dann im Projektordner folgendermaßen aus einer `.rst`-Datei eine gleichnamige `.tex`-Datei sowie das zugehörige `.pdf`-Dokument erzeugen:

```
# RST-Datei dateiname.rst konvertieren:
# (Dateiendung dabei weglassen!)
make -f pfad-zur-rstmakefile dateiname

# Ergebnis: dateiname.tex, dateiname.pdf
```

Diese Methode hat zwei sehr schöne Nebeneffekte: Erstens wird, anders als bei Verwendung von Sphinx, ein „klassischer“ LaTeX-Code ohne Extra-Konfigurationen und besonderen Stil-Elementen generiert. Zweitens können über die Konfigurationsdatei `~/.docutils` optional zusätzliche Pakete in die LaTeX-Präambel geladen werden, um beispielsweise das Seitenlayout anzupassen. Meine Konfigurationsdatei hat beispielsweise folgenden Inhalt:

```
[latex2e writer]
latex_preamble: \usepackage{units,amsmath,amsfonts,amssymb,textcomp,gensymb,
↪marvosym,wasysym}
                \usepackage[left=2cm,right=2cm,top=1cm,bottom=1.5cm]{geometry}
                \setlength{\parskip}{\baselineskip} % Extra line between␣
↪paragraphs
                \setlength{\parindent}{0pt} % No indent at the start of␣
↪paragraphs
                \pagestyle{empty}
```

Durch diese Einstellungen werden einerseits Zusatz-Pakete für mathematischen Formelsatz eingebunden, zum anderen werden durch das `geometry`-Paket die Seitenränder auf ein Minimum reduziert, so dass beim Drucken einzelner Notiz-Seiten kein Platz verschwendet wird.

ReStructuredText-Tutorial

RestructuredText (RST) ist eine vereinfachte Auszeichnungssprache („markup language“), mit deren Hilfe Textdokumente aus einfachen Strukturmerkmalen erzeugt werden können. Diese lassen sich dann mittels eines geeigneten Converters, z.B. Sphinx, automatisiert in LaTeX- oder HTML-Code übersetzen.

Um eine Datei mit RestructuredText-Inhalten zu erstellen, genügt es eine neue Textdatei mit einem Editor zu öffnen. Üblicherweise werden RestructuredText-Dateien mit der Endung `.rst` versehen.

Überschriften und Hervorhebungen

Überschriften sind zur Untergliederung eines Dokuments hilfreich und ermöglichen eine automatische Erzeugung von Inhaltsverzeichnissen. RestructuredText kennt dabei folgende Unterteilungen. *Beispiel:*

```
Name eines Kapitels
===================

Name eines Abschnitts
---------------------

Name eines Unterabschnitts
^^^^^^^^^^^^^^^^^^^^^^^^^^

Name eines Paragraphen
''''''''''''''''''''''
```

Ergebnis:

Zur Hervorhebung einer Überschrift muss somit nur die jeweilige Zeile durch eine zweite, ebenso lange Zeile mit entsprechenden Zeichen markiert werden; anschließend muss eine leere Zeile folgen.

Die Überschriften werden – je nach Converter und entsprechenden Einstellungen – in den erzeugten HTML- beziehungsweise PDF-Dateien mit unterschiedlicher Schriftgröße und/oder nummeriert dargestellt. In PDF-Dateien wird automatisch ein Inhaltsverzeichnis zu Beginn des Dokuments erzeugt, in HTML-Dateien wird das Inhaltsverzeichnis üblicherweise in der Seitenleiste angezeigt.

Zusätzlich ist es möglich, kleine Überschriften ohne Nummerierung und Auflistung im Inhaltsverzeichnis einzufügen. Hierfür wird eine so genannte „Direktive“ namens `.. rubric::` verwendet.[1]

Beispiel:

```
Kapitelname
===========

Text.


.. rubric:: Bezeichnung

Mehr Text..
```

Eine weitere Aufgliederung eines Kapitels ist durch ein Aufspalten in mehrere Dateien möglich.

Das Inhaltsverzeichnis

Üblicherweise werden für die einzelnen Kapitel einer Dokumentation eigene Dateien angelegt, deren Namen in etwa den jeweiligen Kapitelüberschriften entsprechen sollten (auf Umlaute im Dateinamen sollte dabei verzichtet werden, Leerzeichen durch Binde- oder Unterstriche ersetzt werden). Jede dieser Dateien muss unmittelbar mit einer Kapitelüberschrift beginnen und kann weitere Überschriften beinhalten.

In der Hauptdatei `index.rst` der Dokumentation, die im Hauptverzeichnis zu finden ist, wird auf die einzelnen Kapitel über ein Inhaltsverzeichnus verwiesen. Um ein Inhaltsverzeichnis zu erzeugen, wird die so genannte „toctree“-Umgebung verwendet („Table-of-Contents“). Die Syntax dabei ist folgende:

```
.. toctree::
    :maxdepth: 2

    kapitelname-1.rst
    kapitelname-2.rst
    kapitelname-3.rst
```

[1] Eine „Direktive“ ist ein Syntax-Element, das Auswirkung auf einen ganzen Absatz hat, also auf einen Bereich, der durch zwei leere Zeilen begrenzt wird. Im kürzesten Fall, wie bei der `.. rubric::`-Direktive, besteht der Absatz aus einer einzelnen Zeile, die unmittelbar hinter dem Namen der Direktive angegeben wird.

Eine Direktive wird allgemein durch zwei Punkte und ein Leerzeichen zu Beginn einer Zeile eingeleitet, gefolgt vom Namen der Direktive, zwei Doppelpunkten und einem Leerzeichen: `.. name::` . Vor und nach einer Direktive muss (mindestens) eine Leerzeile eingefügt werden.

Je nach Art der Direktive kann hinter ihrem Namen eine weitere Bezeichnung und/oder eine beliebige Anzahl von Absätzen folgen. Um den Wirkungsbereich der Direktive kenntlich zu machen, werden die Absätze dabei eine Tabulatorbreite weit eingerückt (üblicherweise 4 Leerzeichen).

Siehe auch Liste aller RST-Direktiven (en.).

Durch die Option `:maxdepth:` wird festgelegt, bis zu welcher Hierarchie-Stufe das Inhaltsverzeichnis aufgegliedert werden soll. Mit `:maxdepth: 2` werden beispielsweise alle Kapitel- und Abschnittsnamen eingeblendet, die in den angegebenen Dateien zu finden sind. In der HTML-Version wird für jede `toctree` angegebene Datei eine neue Seite inklusive Rahmen und Navigationshilfen erzeugt. In der LaTeX-Version werden die im `toctree` angegeben Dateien der Reihe nach eingebunden, als ob sich ihre Inhalte hintereinander in einer einzigen Datei befänden.

Ein besonderer Vorteil dieser Methode liegt darin, dass umfangreichere Kapitel beliebig in weitere Unterkapitel aufgeteilt werden können:

- Als erstes wird ein neuer Ordner angelegt, der den gleichen Namen wie die zugehörige RestructuredText-Datei erhält, beispielsweise `kapitelname-2`.
- Anschließend wird die Kapitel-Datei in den neuen Ordner verschoben und dort in `index.rst` umbenannt.
- Für jedes Unterkapitel wird in dem neuen Ordner eine neue Textdatei angelegt, deren Namen wiederum in etwa den Überschriften der einzelnen Abschnitte entsprechen sollten. Jeder Abschnitt wird dann aus der `index.rst` ausgeschnitten und in die entsprechende Datei eingefügt.
- In die `index.rst` wird schließlich ein `toctree` angelegt, der die Namen aller Dateien, aus denen das Kapitel besteht, beinhaltet.

Von welcher Hierarchie-Ebene die Überschriften in den einzelnen Dateien ausgehen, ist nicht von Bedeutung: Beim Konvertieren der Quelltexte nach HTML oder PDF werden alle Hierarchie-Ebenen bei Bedarf automatisch angepasst. Es muss lediglich innerhalb jeder Datei darauf geachtet werden, dass unmittelbar mit der „höchste“ Überschrifts-Ebene begonnen wird. Eine Empfehlung hierfür ist, jede Datei mit einer Kapitelüberschrift zu beginnen und bei Bedarf weitere Überschriften einzufügen.

Kommentare

RestructuredText-Dateien können um Kommentare ergänzt werden, die bei der Übersetzung in PDF- beziehungsweise HTML-Dateien ignoriert werden und somit lediglich als „private“ Notizen für den Autor dienen.

Jede Zeile einer RST-Datei kann, indem zu Beginn zwei Punkte und (mindestens) ein Leerzeichen eingefügt werden, zu einem Kommentar gemacht werden.

Beispiel:

```
..  Dies hier ist ein Kommentar.
```

Um einen längeren, aus mehreren Zeilen bestehenden Kommentar zu erzeugen, kann einerseits jede Zeile einzeln durch zwei Punkte und ein Leerzeichen am Anfang der Zeile auskommentiert werden. Einfacher ist es, einen „langen Kommentar“ durch eine separate Zeile einzuleiten, die nur aus zwei Punkten und zwei Leerzeichen besteht:

Beispiel:

```
..
    Dies hier ist ein langer Kommentar.
    Er besteht aus mehreren Zeilen.
```

Auf diese Weise können auch mehrere Absätze auskommentiert werden. Hierbei muss jedoch in den Leerzeilen zwischen den Kommentar-Absätzen Leerzeichen oder Tabulatoren eingefügt werden, da lange Kommentare stets durch eine einzelne, komplett leere Zeile abgeschlossen werden.

Hervorhebung von Textstellen

Um eine einzelne Textstelle innerhalb eines Absatzes hervorzuheben, kann eine so genannte „Role“ verwendet werden.[2] Die wohl am häufigsten auftretenden Roles sind:

- `*Kursiver Text*`: Eine Textstelle, die unmittelbar (ohne Leerzeichen) durch je ein Sternchen begrenzt ist, wird *kursiv* dargestellt.
- `**Fetter Text**`: Eine Textstelle, die unmittelbar (ohne Leerzeichen) durch je zwei Sternchen begrenzt ist, wird **fettgedruckt** dargestellt.
- ``` ``Maschinenschrift`` ```: Eine Textstelle, die unmittelbar (ohne Leerzeichen) durch je zwei schräge Apostrophen („Backticks“) begrenzt ist, wird in `Maschinenschrift` dargestellt. Diese Art der Hervorhebung kann insbesondere für kurze Codebeispiele genutzt werden.
- ``:sub:`Text` ``: Eine Textstelle, die unmittelbar (ohne Leerzeichen) durch je einen schrägen Apostrophen („Backtick“) begrenzt ist und durch das einleitende Schlüsselwort `:sub:` oder `:subscript:` markiert ist, wird als tiefgestellter Text dargestellt.
- ``:sup:`Text` ``: Eine Textstelle, die unmittelbar (ohne Leerzeichen) durch je einen schrägen Apostrophen („Backtick“) begrenzt ist und durch das einleitende Schlüsselwort `:sub:` markiert ist, wird als tiefgestellter Text dargestellt.

Beispiel:

```
Etwas *kursiv dargestellter*,
etwas **fettgedruckter** Text,
und etwas Text in ``Maschinenschrift``;

Tief gestellter Text: :sub:`123` und
hoch gestellter Text: :sup:`456`
```

[2] Eine „Role“ ist ein Syntax-Element, das Auswirkung auf eine Textstelle innerhalb eines Absatzes hat, d.h. auf einen Bereich, der durch zwei Leerzeichen begrenzt wird („Inline-Markup“).

Eine Role hat im allgemeinen folgende Struktur: ``:name:`Inhalt` ``. Die einzigen Ausnahmen bilden die drei oben genannten (wohl am häufigsten auftretenden) Roles für kursiven und fettgedruckten Text sowie Text in Maschinenschrift. Sie stellen praktisch nutzbare Abkürzungen für ``:emphasis:`Text` ``, ``:strong:`Text` `` sowie ``:literal:`Text` `` dar, um Tippbarbeit zu sparen und den Quelltext lesbarer zu gestalten.

Siehe auch Liste aller RST-Roles (en.).

Mittels der `:math:`-Role können zusätzlich mathematische Formeln, geschrieben als LaTeX-Code, innerhalb einer Zeile eingefügt werden. Beispielsweise liefert `:math:`a^2 + b^2 = c^2`` als Ergebnis die Formel $a^2 + b^2 = c^2$.

Hervorhebung von Absätzen

Um einen Absatz beziehungsweise mehrere Absätze hervorzuheben, kann eine der folgenden Direktiven genutzt werden:

- `.. epigraph::`

  Innerhalb einer `epigraph`-Umgebung werden gewöhnlich Zitate in das Dokument eingefügt. Am Ende wird dabei üblicherweise der Name des Autors der zitierten Textstelle angegeben.

  Beispiel:

  ```
  .. epigraph::

     "Phantasie ist wichtiger als Wissen, denn Wissen ist begrenzt."

     -- Albert Einstein
  ```

  Ergebnis:

  > „Phantasie ist wichtiger als Wissen, denn Wissen ist begrenzt.“
  >
  > —Albert Einstein

  Innerhalb einer `epigraph`-Umgebung sind sowohl mehre Absätze als auch Inline-Markup (Roles) erlaubt. Die Ausgabe erfolgt eingerückt und mit reduzierter Zeilenlänge, um das Zitat gut erkennbar vom übrigen Text abzuheben; Zeilenumbrüche erfolgen automatisch.

- `.. line-block::`

  Die `line-block`-Umgebung ist der `epigraph`-Umgebung ähnlich, jedoch finden im Ergebnis keine automatischen Zeilenumbrüche statt. Die Zeilen werden stattdessen in gleicher Form ausgegeben, wie sie innerhalb der `line-block`-Umgebung gesetzt werden. Dies ist insbesondere beim Zitieren von Gedichten und Versen nützlich:

  Beispiel:

  ```
  .. line-block::

     "Jede Blüte will zur Frucht
     Jeder Morgen Abend werden
     Ewiges ist nicht auf Erden
     Als der Wandel, als die Flucht."

     -- Hermann Hesse (Ausschnitt aus dem Gedicht "Welkes Blatt")
  ```

  Ergebnis:

„Jede Blüte will zur Frucht
Jeder Morgen Abend werden
Ewiges ist nicht auf Erden
Als der Wandel, als die Flucht.“

– Hermann Hesse (Ausschnitt aus dem Gedicht „Welkes Blatt“)

Absätze, die innerhalb einer `line-block`-Umgebung stehen, werden nicht automatisch eingerückt. Ist dies gewünscht, so kann man eine Einrückung entweder über entsprechende CSS-Einstellungen oder über eine manuelle Einrückung der jeweiligen Umgebung (Leerzeichen beziehungsweise Tabulatoren im Quellcode) erreichen.

- `note`, `hint`, `tip`, `warning`, `error`, `important`

  Mit den obigen Direktiven lassen sich Infoboxen erzeugen. Der Titel der Infobox leitet sich dabei aus dem Direktivennamen ab (Bemerkung, Hinweis, Tip, Warnung, Fehler, Wichtig).

  Beispiel:

```
.. hint::

   Hier wird ein Hinweis ausgegeben.
```

  Ergebnis:

Hinweis: Hier wird ein Hinweis ausgegeben.

  Neben den oben genannten Direktiven kann auch eine `topic`-Umgebung genutzt werden, um eine beliebig benannte Infobox erzeugen. Dabei wird in der gleichen Zeile de im Anschluss an `.. topic::` der Name der Box geschrieben.

- `math`

  Mit der `math`-Direktive können mathematische Formeln, geschrieben als LaTeX-Code, als eigenständige zentrierte Zeilen in das Dokument eingebunden werden.

  Beispiel:

```
.. math::

   a^2 + b^2 = c^2
```

  Ergebnis:

$$a^2 + b^2 = c^2$$

  Die `math`-Direktive bietet zusätzlich die Option, der angegebenen Formel eine Sprungmarke („Label“) und eine automatisch vergebene Nummer zu vergeben. Hierzu wird eine eigene Zeile der Form `:label: Name-des-Labels` unmittelbar als erste

Zeile der `math`-Direktive eingefügt (mit gleicher Einrückung wie die eigentliche Formel).

Beispiel:

```
.. math::
    :label: einstein-und-pythagoras

    E \underset{Einstein}{=} m \cdot c^2
    \underset{Pythagoras}{=} m \cdot (a^2 + b^2)
```

Ergebnis:

$$E \underset{Einstein}{=} m \cdot c^2 \underset{Pythagoras}{=} m \cdot (a^2 + b^2) \tag{1}$$

Auf die Formel kann dann mittels der Referenz `eqr:`Name-des-Labels`` an einer beliebigen anderen Stelle des Dokuments (derzeit jedoch nur innerhalb einer einzelnen Quellcode-Datei) verwiesen werden.

- `code-block`

  Die `code-block`-Direktive ermöglicht es, wie der Name bereits andeutet, Quellcode-Beispiele in das Dokument einzufügen. Dabei kann in der gleichen Zeile im Anschluss an `.. code-block::` eine Codesprache aus dieser Liste ausgewählt werden, um ein Syntax-Highlighting zu aktivieren.

  Beispiel:

```
.. code-block:: sh
    :linenos:

    # Show the local network address
    # Result: Something like 192.168.1.105
    hostname -I | cut -d' ' -f1
```

  Ergebnis:

```
1 # Show the local network address
2 # Result: Something like 192.168.1.105
3 hostname -I | cut -d' ' -f1
```

Quellcode wird üblicherweise in Maschinenschrift ausgegeben; jegliches Inline-Markup wird dabei ignoriert. Möchte man Inline-Markup (Roles) dennoch interpretiert haben, um beispielsweise Verlinkungen innerhalb des Quellcodes zu setzen, kann anstelle von `code-block` die `parsed-literal`-Direktive verwendet werden, die ansonsten die gleiche Syntax aufweist.

Weitere Gestaltungsmöglichkeiten von Absätzen sind in der Liste aller RST-Direktiven (en.) aufgeführt.

Aufzählungen und Beschreibungen

In RestructuredText sind sowohl nummerierte wie auch nicht nummerierte Aufzählungen möglich. Die Syntax hierfür ist sehr simpel:

```
*Beispiel einer nummerierten Liste:*

1. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text
   liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der
   Schrift an.
2. | Dies hier ist ein Blindtext zum Testen von Textausgaben.
   | Wer diesen Text liest, ist selbst schuld.
   | Der Text gibt lediglich den Grauwert der Schrift an.

*Beispiel einer nicht nummerierten Liste:*

* Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text
  liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der
  Schrift an.
* | Dies hier ist ein Blindtext zum Testen von Textausgaben.
  | Wer diesen Text liest, ist selbst schuld.
  | Der Text gibt lediglich den Grauwert der Schrift an.
```

Ergebnis:

Beispiel einer nummerierten Liste:

1. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an.
2. Dies hier ist ein Blindtext zum Testen von Textausgaben.
   Wer diesen Text liest, ist selbst schuld.
   Der Text gibt lediglich den Grauwert der Schrift an.

Beispiel einer nicht nummerierten Liste:

- Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an.
- Dies hier ist ein Blindtext zum Testen von Textausgaben.
  Wer diesen Text liest, ist selbst schuld.
  Der Text gibt lediglich den Grauwert der Schrift an.

Für nicht nummerierte Aufzählungen können anstelle des Zeichens `*` auch die Zeichen `-` oder `+` verwendet werden; in der Ausgabe werden die Aufzählungszeichen unabhängig davon anhand der Aufzählungstiefe ausgewählt (innerhalb einer Aufzählung sind auch weitere Aufzählungen möglich).

Beim Erstellen von Aufzählungen muss lediglich darauf geachtet werden, dass vor und nach der Aufzählung im Quellcode eine leere Zeile steht und die einzelnen Einträge gleich weit eingerückt sind. Die einzelnen Einträge werden, sofern sie nicht in eine Zeile passen,

automatisch am Seitenrand umgebrochen; manuelle Umbrücke können erzwungen werden, indem zu Beginn jeder neu zu erstellenden Zeile ein `|`-Zeichen eingegeben wird. Auch hierbei muss auf eine gleiche Einrückungstiefe der Textzeilen geachtet werden.

Automatisch nummerierte Aufzählungen können mittels `#.` als Aufzählungszeichen erstellt werden; in diesem Fall sind zwischen den einzelnen Einträgen allerdings keine RestructuredText-Elemente erlaubt, die ohne Einrückung zu Beginn der Zeile eingegeben werden müssen (beispielsweise Sprungmarken); in diesem Fall fängt die folgende Aufzählung nämlich wieder mit `1.` an.

Bei Beschreibungen verwendet man das zu beschreibende Wort als „Aufzählungszeichen“. Die Syntax hierzu ist folgende:

```
**Wort1:**
    Beschreibung zu Wort1.

**Wort2:**
    Beschreibung zu Wort2.
```

Ergebnis:

Wort1: Beschreibung zu Wort1.

Wort2: Beschreibung zu Wort2.

Auch bei den Beschreibungen muss nur auf die einheitliche Einrückung des Beschreibungs-Textes geachtet werden; der Text kann dann auch aus mehreren Absätzen bestehen.

Sprungmarken und Referenzen

Ein sehr nützliche von Wiki-Seiten besteht darin, mittels eines klickbaren Links auf eine andere Stelle in der Dokumentation oder auf eine externe Seite verweisen zu können. RestructuredText bietet dazu folgende Möglichkeiten:

- Mittels `` `Link-Bezeichnung <Adresse>`_ `` kann ein mit einer bestimmten Bezeichnung versehener Link auf eine externe Seite gesetzt werden. Soll eine Adresse ohne eigene Bezeichnung verlinkt werden, so genügt es die Adresse ohne weitere Syntax anzugeben, beispielsweise `http://www.grund-wissen.de` für http://www.grund-wissen.de ; ein Link wird dabei automatisch erzeugt.
- Mittels einer eigenen Zeile der Form `.. _Name der Sprungmarke:` und einer darauf folgenden Leerzeile kann an einer beliebigen Stelle innerhalb der Dokumentation eine Sprungmarke (auch „Label“ oder „Anker“ genannt) festgelegt werden. Auf diese Sprungmarke kann dann von einer beliebigen anderen Stelle im Dokument aus mittels `:ref:`Link-Bezeichnung <Name der Sprungmarke>`` verwiesen werden.

  Diese Methode funktioniert auch, wenn sich die Sprungmarke und die Referenz in verschiedenen Dateien des Quelltextes einer Dokumentation befinden.

- Mittels der „Intersphinx“-Erweiterung, die beim Sphinx-Quickstart ausgewählt werden kann[3], ist es möglich, auch auf Sprungmarken anderer Sphinx-Projekte zu verweisen. Hierzu muss die Konfigurationsdatei `conf.py` um einen oder mehrere Einträge mit folgender Form ergänzt werden:

```
intersphinx_mapping = {
    'sphinx': ('http://sphinx-doc.org', None),
    'gw': ('http://grund-wissen.de', None)
}
```

  Damit kann beispielsweise mittels `:ref:`Inhaltsverzeichnis der Sphinx-Dokumentation <sphinx:contents>`` ein Link auf das Inhaltsverzeichnis der Sphinx-Dokumentation gesetzt werden, das eine Sprungmarke namens `contents` enthält (dies zeigt ein Blick in den Quelltext der Seite, der üblicherweise in der Seitenleiste verlinkt ist). Führt man den Mauszeiger über einen solchen Link, so werden der Name und die Versionsnummer der jeweiligen Dokumentation eingeblendet.

Fußnoten, Zitierungen und Index-Einträge

In RestructuredText gibt es die Möglichkeit, ergänzende Anmerkungen als Fußnoten aus dem normalen Text „auszulagern“. Hierzu wird im Haupttext eine Marke der Form `[#]_` oder `[#Name]_` gesetzt, d.h. ein Rautenzeichen in eckigen Klammern, gefolgt von einem Unterstrich.[4] Optional kann jeder Marke einer Fußnote im Anschluss an die Nummer oder das Rautensymbol (Autonummerierung) noch ein Name hinzugefügt werden, um im Quelltext die Zuordnung der Fußnoten-Marke zur Fußnote zu erleichtern.

An einer späteren Stelle innerhalb der gleichen Datei, meist am Ende, wird der Inhalt der jeweiligen Fußnote dann absatzweise mittels `.. [#] Inhalt` beziehungsweise `.. [#Name] Inhalt` angegeben, wobei `[#Name]` der Marke im Haupttext entsprechen muss.

Beispiel:

```
Etwas Text. [#FN1]_

Weiterer Text.

...


.. [#FN1] Eine Anmerkung als Fußnote.
```

[3] Die Intersphinx-Erweiterung lässt sich ebenso nutzen, wenn in der Konfigurationsdatei `conf.py` die `extension`-Liste um den Eintrag `'sphinx.ext.intersphinx'` ergänzt wird.

[4] Optional können die Nummern der Fußnoten auch in der Art `[01]_`, `[02]_` beziehungsweise `[01Name]_`, `[02Name]_` usw. selbst vergeben werden. Davon ist allerdings abzuraten, denn sollte zu einem späteren Zeitpunkt an einer Stelle mitten im Text eine weitere Fußnote eingefügt werden, so müssen die Nummern aller folgenden Fußnoten manuell angepasst werden. Durch automatisch nummerierte Fußnoten bleibt einem diese Arbeit sicher erspart.

Tip: Durch die Option `trim_footnote_reference_space = True` in der `conf.py` wird ein mögliches Leerzeichen vor Fußnoten, wie in deutschsprachiger Literatur üblich, ignoriert.

Ergebnis:

Erstreckt sich der Inhalt einer Fußnote über mehrere Zeilen, so muss jede Zeile nach der ersten um mindestens ein Leerzeichen eingerückt werden (üblicherweise werden Folgezeilen eine Tabulatorbreite weit eingerückt, um eine bessere Lesbarkeit zu erzielen).

Zitierungen und Literaturverzeichnis

Innerhalb einer Dokumentation sind auch Verweise auf literarische Werke anderer Autoren möglich. Für jedes zitierte Werk wird dabei ein Kurzname vergeben, häufig in der Form `AutorJahr`. Im Haupttext (oder in einer Fußzeile) kann auf diese Weise mittels `[Kurzname]_` auf eine genauere Umschreibung der Literaturquelle verwiesen werden, die einmalig an einer beliebigen Stelle der Dokumentation mittels eines Eintrags der Form `.. [Kurzname] Informationen` erfolgt.[5]

In der HTML-Version werden alle Literatur-Einträge an genau der Stelle eingefügt, an der sie gesetzt werden. Insofern empfiehlt sich eine eigene Datei namens `quellen.rst` (oder ähnlich), in der die Literaturhinweise und Quellenangaben gesammelt aufgelistet sind. In der LaTeX-Version wird am Ende des Dokuments automatisch ein Literaturverzeichnis angelegt.

Index-Einträge

Innerhalb der Dokumentation können an beliebiger Stelle mittels folgender Syntax Einträge für ein Stichwortverzeichnis festgelegt werden:

```
.. index:: Bezeichnung
```

In der HTML-Version wird ein Link auf die Index-Seite üblicherweise auf der rechten Seite am oberen und unteren Seitenrand eingeblendet. In der LaTeX-Version wird das Stichwortverzeichnis auf den letzten Seiten der Dokumentation abgedruckt. Eine Verlinkung mit den entsprechenden Textstellen (in der Druckversion mitsamt Angabe der jeweiligen Seitennummer) erfolgt automatisch.

- Um mehrere Index-Einträge zur gleichen Textstelle zu erreichen, können die Bezeichnungen der gewünschten Einträge, durch Kommas voneinander getrennt, in einer einzigen Zeile aufgelistet werden:

  ```
  .. index:: Eintrag1, Eintrag2
  ```

- Werden zwei Einträge durch einen Strichpunkt getrennt, so wird der zweite Eintrag als „Unterkategorie“ des ersten im Stichwortverzeichnis angezeigt:

[5] Die Syntax von Zitierungen ähnelt somit der Syntax von Fußnoten, mit dem Unterschied, dass innerhalb der eckigen Klammern keine Nummer beziehungsweise kein einleitendes Raute-Zeichen auftritt.
Jede Literaturangabe sollte folgende Informationen beinhalten: Name des Autors beziehungsweise der Autoren, Titel des Werks, (gegebenenfalls) Name des Verlags, Erscheinungsjahr.

```
.. index:: Eintrag; Unterkategorie
```

Eine ausführliche Beschreibung findet sich in der Sphinx-Dokumentaion.

Bilder und Tabellen

Bilder können in Restructured-Text entweder mittels einer `image`- oder mittels einer `figure`-Direktive eingebunden werden; letztere muss verwendet werden, wenn die Abbildung eine Bildunterschrift („Caption“) erhalten soll.

Die Syntax für den Einbau eines Bildes sieht etwa folgendermaßen aus:

```
.. figure:: ../pics/no-littering.png
    :name: fig-beispiel-bild
    :alt:  fig-beispiel-bild
    :align: center
    :width: 20%

    Beispiel-Bild "No littering" ohne weitere Beschreibung.

    .. only:: html

        :download:`SVG: Beispiel-Bild ("No littering") <../pics/no-littering.
↪svg>`
```

Ergebnis:

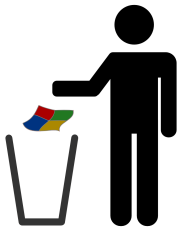


Abb. 2: Beispiel-Bild „No littering“ ohne weitere Beschreibung.

Bei Verwendung der `figure`-Direktive wird zunächst der Pfad der einzubindenden Graphik angegeben. Die `:name:`-Attribut angegebene Bezeichnung kann in Referenzen als Label aufgegriffen werden; die als `:alt:`-Attribut angegebene Bezeichnung wird im Webbrowser während des Ladens der Graphik oder im Fall einer nicht auffindbaren Graphik-Datei angezeigt.

Über das Attribut `:align:` wird die Ausrichtung der Graphik am Text als `left`, `center` oder `right` festgelegt. Die Angabe der Breite kann über das `:width:`-Attribut entweder als feste Längeneinheit (beispielsweise `4.2 cm`) oder relativ zur Textbreite als Prozentangabe erfolgen. Als Alternative zur Angabe einer Breite kann die Größe einer Abbildung auch mittels einer `:height:`- oder `scale`-Angabe (in Prozent der Original-Bildgröße) festgelegt werden. Bei der Angabe einer Bildunterschrift innerhalb einer `figure`-Umgebung muss wiederum auf die richtige Einrücktiefe geachtet werden.

Bei der `image`-Direktive können, abgesehen von der Bildunterschrift, ebenfalls die oben genannten Attribute angegeben werden; dafür kann (nur) bei der `image`-Direktive als zusätzliches Argument mittels `:target:` eine Zieladresse angeegben werden, auf die beim Anklicken des Bildes verlinkt wird.

Zur Eingabe von Tabellen gibt es in RestructuredText ebenfalls mehrere Möglichkeiten:

- Bei einer „Gitter-Tabelle“ wird die tabellarische Anordnung bereits im Quellcode angedeutet:

```
+------------------+--------------+---------------+
| Art der Einträge | Eigenschaft 1 | Eigenschaft 2 |
+==================+===============+===============+
| Gegenstand 1     | eckig         | rot           |
+------------------+--------------+---------------+
| Gegenstand 2     | rund          | blau          |
+------------------+--------------+---------------+
```

  Ergebnis:

Art der Einträge	Eigenschaft 1	Eigenschaft 2
Gegenstand 1	eckig	rot
Gegenstand 2	rund	blau

  Werden beim ersten „Querstrich“ in der Tabelle anstelle der =-Zeichen --Zeichen gesetzt, so wird die erste Zeile in normaler Schrift (nicht fettgedruckt) ausgegeben.

  Tabellen gemäß dieser Syntax sind mühsam, sofern der Texteditor nicht ein geeignetes Feature dafür mitbringt (für den Editor *Vim* gibt es hierfür beispielsweise das *Table-Mode*-Plugin). Der Vorteil solcher Tabellen liegt vor allem in der angenehmen Lesbarkeit im Quellcode.

  Nicht geeignet sind Grid-Tabellen, wenn auch Links oder mathematische Formeln in der Tabelle vorkommen sollen. Diese machen im Quellcode mehr Buchstaben aus als in der Ausgabe, so dass die Spalten dadurch breiter als nötig gedruckt werden. Hierfür sollte eher „List-Tables“ verwendet werden.

- Bei „List Tables“ werden die Einträge einer Tabelle in Form einer verschachtelten Aufzählung angegeben. Die Syntax hierzu sieht etwa folgendermaßen aus:

```
.. list-table::
    :name: tab-beispieltabelle
    :widths: 50 50 50
    :header-rows: 0

    * - Art der Einträge
      - Eigenschaft 1
      - Eigenschaft 2
    * - Gegenstand 1
      - eckig
      - rot
    * - Gegenstand 2
```

```
      - rund
      - blau
```

Ergebnis:

Art der Einträge	Eigenschaft 1	Eigenschaft 2
Gegenstand 1	eckig	rot
Gegenstand 2	rund	blau

Soll die erste Zeile der Tabelle fett gedruckt ausgegeben werden, muss das Attribut `:header-rows` auf `1` gesetzt werden.

Der Vorteil von List-Tables liegt darin, dass in den einzelnen Einträgen beliebige Inline-Direktiven verwendet werden können (Links, Formeln, usw). Ebenso kann die „Gewichtung“ der einzelnen Spaltenbreiten durch die Angabe der `:widths:`-Werte manuell angepasst werden. Im obigen Beispiel werden die drei Spalten der Tabelle in einem Breitenverhältnis von `50:50:50`, also mit gleicher Breite ausgegeben.

Wird in der gleichen Zeile direkt hinter `.. list-table::` ein Titel angegeben, so wird dieser als Überschrift über die Tabelle gedruckt.

- Mit einer „CSV-Table“ kann der Inhalt der Tabelle durch „Comma Seperated Values“ erfolgen. Diese Einträgte können wahlweise im Block der Listen-Direktive oder in einer separaten `.csv`-Datei angegeben werden. Das obige Beispiel kann damit etwa folgendermaßen aussehen:

```
.. csv-table::
    :widths: 50 50 50

    Art der Einträge , Eigenschaft 1 , Eigenschaft 2
    Gegenstand 1 , eckig , rot
    Gegenstand 2 , rund , blau
```

Ergebnis:

Art der Einträge	Eigenschaft 1	Eigenschaft 2
Gegenstand 1	eckig	rot
Gegenstand 2	rund	blau

Soll anstelle von `,` ein anderes Zeichen zur Trennung der einzelnen Einträge verwendet werden, so kann dieses mittels des `:delim:`-Attributs angegeben werden.

Anstelle einer direkten Eingabe der Tabelle kann auch der Inhalt einer externen `.csv`-Datei verwendet werden; hierzu muss entweder ein Pfad als `:file:`-Attribut oder eine Web-Adresse als `:url:`-Attribut angegeben werden.

Substitutionen

Mittels Substitutionen kann innerhalb eines Absatzes ein Text-Label durch einen zugehörigen anderen Inhalt ersetzt werden. Dies ist insbesondere hilfreich, um kleine Graphiken wie Icons in den Fließtext einzubinden; dies ist insbesondere bei der Dokumentation von Programmen mit graphischen Bedienoberflächen nützlich.

Die Syntax hierfür lautet etwa folgendermaßen:

Beispiel:

```
.. Variante 1: Text-Hervorherbung und Ersetzung

Blindtext |hallo-welt| Blindtext

.. Variante 2: Einbau einer Icon-Graphik

Blindtext |symbol| Blindtext


.. |hallo-welt| replace:: **Hallo Welt!**

.. |symbol| image:: pics/symbol-amperemeter-klein.png
                .. alt: Amperemeter-Symbol
```

Ergebnis:

Blindtext **Hallo Welt!** Blindtext

Blindtext  Blindtext

Substitution können also, wie das obige Beispiel zeigt, weitgehend ähnlich wie Fußnoten verwendet werden, mit dem wesentlichen Unterschied, dass |-Symbole anstelle der eckigen Klammern gesetzt werden müssen.

Links

- A RestructuredText Primer (en.)
- Quick RestructuredText (en.)
- RestructuredText and Sphinx CheatSheet (en.)

Gestalterische Anpassungen

Für die Grundwissen-Webseite nutze ich das Sphinx-Standard-Theme mit einigen Anpassungen. Diese habe ich als Super-User direkt am zentralen Installationspfad (in meinem Fall `/usr/local/lib/python3.5/dist-packages/Sphinx-1.4.5-py3.5.egg/sphinx`) vorgenommen, so dass sie global für alle Dokumentations-Projekte gleichermaßen

gelten. Als (empfehlenswertere) Alternative hierzu kann Sphinx mittels einer virtuellen Umgebung lokal im Ordner des Benutzers installiert und dort auch ohne Super-User-Rechte angepasst werden. Die im folgenden Abschnitt aufgelisteten Datei-Änderungen beziehen sich dabei stets auf den jeweiligen Installationspfad.

Kontakt-Link auf jeder HTML-Seite hinzufügen

In der Fußnotenzeile der Webseite wollte ich ein Link auf Kontakt/Impressum eingefügen. Dies kann in in der Datei `themes/basic/layout.html` festgelegt werden:

```
VORHER:

{%- if last_updated %}
  {% trans last_updated=last_updated|e %}Last updated on {{ last_updated }}.
  {% endtrans %}
{%- endif %}

NACHHER:

{%- if last_updated %}
  {% trans last_updated=last_updated|e %}
  Zuletzt aktualisiert am {{ last_updated }}.
  <a href='http://grund-wissen.de/impressum.html'>Kontakt/Impressum</a>
  {% endtrans %}
{%- endif %}
```

Zeilenumbruch bei langen HTML-Navigationszeilen ermöglichen

In der obersten Zeile einer jeden mit Sphinx erstellten HTML-Seite wird eine Navigations-Leiste angezeigt. Bei einer umfangreichen Dokumentation mit vielen Unterabschnitten kann es vorkommen, dass auf kleinen Bildschirmen hierbei ein Zeilenumbruch nötig ist – der letzte Listeneintrag wird also in eine neue Zeile geschrieben. In der Grundversion wird hierbei die Seitenüberschrift verschoben. Um dies zu vermeiden, muss folgender Eintrag in der Datei `themes/basic/static/basic.css_t` ergänzt werden:

```
VORHER:

div.related ul {
    margin: 0;
    padding: 0 0 0 10px;
    list-style: none;
    }


NACHHER:

div.related ul {
    margin: 0;
```

```
    padding: 0 0 0 10px;
    list-style: none;
    min-height: 2em;
    height: auto;
    overflow: hidden;
    }
```

Durch den Eintrag `height: auto` wird die Höhe der Navigations-Leiste automatisch angepasst. Der Eintrag `overflow: hidden;` fügt anschließend bei Bedarf automatisch eine (wieder ganz von links beginnende) neue Zeile ein.

Maximalbreite der HTML-Dokumentation festlegen

In den letzten Jahren sind Computer-Monitore immer breiter geworden. Im Vergleich zu einer DinA4-Seite sind somit, sofern keine Beschränkungen festgelegt werden, auch wesentlich längere Textzeilen möglich.

Mittels einer entsprechenden CSS-Option kann erreicht werden, dass auch bei sehr breiten Monitoren ein als kleiner Absatz geschriebener Text als eine einzige Zeile erscheint – das Layout der HTML-Seite bleibt somit der Druckversion ähnlich. Dies gilt nicht nur für Text, sondern auch für Graphiken, die mit einer prozentualen Breitenangabe eingebunden werden: In LaTeX ist die Zeilenbreite des Texts als Referenzwert üblich, in HTML standardmäßig die Breite des Browserfensters; auf breiten Monitoren werden Graphiken in einem Vollbild-Browser entsprechend groß dargestellt.

Bei der Grund-Wissen-Seite wird das Theme `sphinxdoc` verwendet; die entsprechende CSS-Datei ist `themes/sphinxdoc/static/sphinxdoc.css_t`. Hier muss zur Begrenzung der HTMl-Seitenbreite folgende Änderung vorgenommen werden:

```
NACHHER:

body {
    font-family: 'Lucida Grande', 'Lucida Sans Unicode', 'Geneva',
                 'Verdana', sans-serif;
    font-size: 14px;
    letter-spacing: -0.01em;
    line-height: 150%;
    text-align: center;
    background-color: #BFD1D4;
    color: black;
    padding: 0;
    border: 1px solid #aaa;

    margin: auto;
    min-width: 740px;
    max-width: 1200px;
}
```

Mit den obigen Einstellungen wird die Webseite mit der angegebenen Breite zentriert dargestellt; der Rest des Browserfensters durch einen neutralen Hintergrund ausgefüllt.

Tabellen zentrieren

Tabellen werden von Sphinx bei Verwendung der Grundeinstellungen automatisch zentriert; Persönlich möchte ich in der HTML-Darstellung allerdings die Fußnoten, die via CSS ebenfalls als Tabellen gesetzt werden, linksbündig ausrichten. Um dies zu erreichen, habe ich in der Datei `themes/basic/static/basic.css_t` den Eintrag `table.docutils` folgendermaßen ergänzt:

```
table.docutils {
    border: 1px solid gray;
    border-collapse: collapse;
    margin-left: auto;
    margin-right: auto;
}

table.docutils.footnote, table.docutils.citation {
    border: 0px;
    border-collapse: collapse;
    margin-left: 0;
}

table.docutils.footnote td, table.docutils.citation td {
    border: 0px;
}
```

Kopfzeile in der Druckversion entfernen

Normalerweise erscheint in der mittels LaTeX erzeugten PDF-Datei in der Kopfzeile jeder Seite ein Versionshinweis. Um dies zu deaktivieren, muss die Datei `texinputs/sphinx.sty` an zwei Stellen abgeändert werden. Zum einen muss die `\fancyhead`-Zeile, die den Versions-Eintrag erzeugt, mit einem `%`-Zeichen auskommentiert werden. Zum anderen kann der Strich zwischen Kopfzeile und erster Textzeile entfernt werden, indem ihr Wert auf `0.0` gesetzt wird:

```
VORHER:

\fancypagestyle{normal}{
  \fancyhf{}
  \fancyfoot[LE,RO]{{\py@HeaderFamily\thepage}}
  \fancyfoot[LO]{{\py@HeaderFamily\nouppercase{\rightmark}}}
  \fancyfoot[RE]{{\py@HeaderFamily\nouppercase{\leftmark}}}
  \fancyhead[LE,RO]{{\py@HeaderFamily \@title, \py@release}}
  \renewcommand{\headrulewidth}{0.4pt}
  \renewcommand{\footrulewidth}{0.4pt}
```

```
NACHHER:

\fancypagestyle{normal}{
  \fancyhf{}
  \fancyfoot[CE,CO]{{\py@HeaderFamily\thepage}}
% \fancyfoot[LO]{{\py@HeaderFamily\nouppercase{\rightmark}}}
% \fancyfoot[RE]{{\py@HeaderFamily\nouppercase{\leftmark}}}
% \fancyhead[LE,RO]{{\py@HeaderFamily \@title, \py@release}}
  \renewcommand{\headrulewidth}{0.0pt}
  \renewcommand{\footrulewidth}{0.4pt}
```

Mehrspaltige Aufzählungen (hlist) in LaTeX

Mit der `hlist`-Umgebung kann man mit Sphinx mehrspaltige Tabellen erstellen. Der Code dafür sieht etwa so aus:

```
.. hlist::
    :columns: 2

    * Item 1
    * Item 2
    * ...
```

Während die HTML-Ausgabe ausgezeichnet funktioniert, werden `hlist`-Umgebungen vom LaTeX-Übersetzer wie „normale“ Listen behandelt. Persönlich verwende ich in den allermeisten Fällen zweispaltige `hlists`, so dass ich mir in der Datei `writers/latex.py` mit folgendem Trick Abhilfe für den erstellten LaTeX-Code geschaffen habe:

```
VORHER:

\usepackage{sphinx}

[...]

def visit_hlist(self, node):
    self.compact_list += 1
    self.body.append('\\begin{itemize}\\setlength{\\itemsep}{0pt}'
                     '\\setlength{\\parskip}{0pt}\n')

[...]

def depart_hlist(self, node):
    self.compact_list -= 1
    self.body.append('\\end{itemize}\n')


NACHHER:
```

```
\usepackage{sphinx}
\usepackage{multicol}

[...]

def visit_hlist(self, node):
    self.compact_list += 1
    self.body.append('\\begin{multicols}{2}')
    self.body.append('\\begin{itemize}\\setlength{\\itemsep}{0pt}'
                     '\\setlength{\\parskip}{0pt}\n')

[...]

def depart_hlist(self, node):
    self.compact_list -= 1
    self.body.append('\\end{itemize}\n')
    self.body.append('\\end{multicols}')
```

Damit werden alle `hlists` in der Druckversion als zweispaltige Aufzählungen dargestellt.[1]

Darstellung von Verbatim-Boxen anpassen

Um Code-Beispiele in LaTeX besser hervorzuheben, habe ich in der Datei `texinputs/sphinx.sty` die Farben für die Verbatim-Umgebung und ihre Umrandung etwas angepasst:

```
VORHER:

\definecolor{VerbatimColor}{rgb}{1,1,1}
\definecolor{VerbatimBorderColor}{rgb}{1,1,1}

NACHHER:

\definecolor{VerbatimColor}{rgb}{0.97,0.97,1}
\definecolor{VerbatimBorderColor}{rgb}{0.75,0.75,1}
```

Die Boxen werden so in einem schwachen Blau mit einem ebenfalls leicht blauen Rahmen gedruckt.

Titelseite gestalten

Nach persönlichem Geschmack habe ich die Titelseite etwas abgewandelt – insbesondere wollte ich dort einen Link auf die URL der Homepage einfügen. Hierbei habe ich in der Datei `texinputs/sphinxmanual.cls` die Funktion `\maketitle` folgendermaßen angepasst;

[1] Hierbei muss das LaTeX-Paket `multicol` installiert sein. Sollte dies nicht der Fall sein, kann es von der CTAN-Projektseite herunter geladen werden und gemäß dem üblichen Installations-Schema nachinstalliert werden.

```
\renewcommand{\maketitle}{%
  \let\spx@tempa\relax
  \ifHy@pageanchor\def\spx@tempa{\Hy@pageanchortrue}\fi
  \hypersetup{pageanchor=false}% avoid duplicate destination warnings
  \begin{titlepage}%
    \let\footnotesize\small
    \let\footnoterule\relax
    \noindent\rule{\textwidth}{1pt}\par
      \begingroup % for PDF information dictionary
       \def\endgraf{ }\def\and{\& }%
       \pdfstringdefDisableCommands{\def\\{, }}% overwrite hyperref setup
       \hypersetup{pdfauthor={\@author}, pdftitle={\@title}}%
      \endgroup
    \begin{flushright}%
      \sphinxlogo
      \py@HeaderFamily
      {\Huge \@title \par}
      {\itshape\LARGE \py@release\releaseinfo \par}
      {\itshape Aktualisiert am \@date \par}
      \vfill
      {\LARGE
        \begin{tabular}[t]{c}
          \@author
        \end{tabular}
        \par}
      \vfill\vfill
      {\Large
      \url{http://www.grund-wissen.de} \par
       \vfill
       \py@authoraddress \par
      }%
    \end{flushright}%\par
    \@thanks
  \end{titlepage}%

  \setcounter{footnote}{0}%
  \let\thanks\relax\let\maketitle\relax
  %\gdef\@thanks{}\gdef\@author{}\gdef\@title{}
  \if@openright\cleardoublepage\else\clearpage\fi
  \spx@tempa
}
```

Zwischen `\end{titlepage}` und `\setcounter{footnote{0}` habe ich zusätzlich folgenden Text hinzugefügt, der bei jeder meiner Dokumentationen auf der zweiten Seite des Dokuments, also unmittelbar vor dem Inhaltsverzeichnis, erscheinen soll:[2]

```
VORHER:
```

[2] Diese Lösung ist nicht elegant, erfüllt aber ihren Zweck. Für Verbesserungshinweise bin ich dankbar. ;-)

```
[...]

\end{titlepage}%

\setcounter{footnote}{0}%
  \let\thanks\relax\let\maketitle\relax
  %\gdef\@thanks{}\gdef\@author{}\gdef\@title{}
  \if@openright\cleardoublepage\else\clearpage\fi
  \spx@tempa
}


NACHHER:

[...]

\end{titlepage}%

Dieses Buch wird unter der
\href{http://de.wikipedia.org/wiki/Creative_Commons}{Creative Commons
License (Version 4.0, by-nc-sa)} ver\"{o}ffentlicht. Alle Inhalte d\"{u}rfen
daher in jedem beliebigen Format vervielf\"{a}ltigt und/oder
weiterverarbeitet werden, sofern die Weitergabe nicht kommerziell ist, unter
einer gleichen Lizenz erfolgt, und das Original als Quelle genannt wird.
Siehe auch:

\href{https://creativecommons.org/licenses/by-nc-sa/4.0}{Erl\"{a}uterung der
    Einschr\"{a}nkung by-nc-sa} \\
\href{https://irights.info/wp-content/uploads/userfiles/CC-NC_Leitfaden_web.pdf}
↪{Leitfaden
    zu Creative-Commons-Lizenzen}

Unabh\"{a}ngig von dieser Lizenz ist die Nutzung dieses Buchs f\"{u}r
Unterricht und Forschung \href{http://dejure.org/gesetze/UrhG/52a.html}{(\S
52a UrhG)} sowie zum privaten Gebrauch
\href{http://dejure.org/gesetze/UrhG/53.html}{(\S 53 UrhG)} ausdr\"{u}cklich
erlaubt.

Der Autor erhebt mit dem Buch weder den Anspruch auf Vollst\"{a}ndigkeit
noch auf Fehlerfreiheit; insbesondere kann f\"{u}r inhaltliche Fehler keine
Haftung \"{u}bernommen werden.

Die Quelldateien dieses Buchs wurden unter
\href{http://grund-wissen.de/linux/index.html}{Linux} mittels
\href{http://grund-wissen.de/linux/tools/vim/index.html}{Vim} und
\href{http://grund-wissen.de/linux/tools/sphinx/index.html}{Sphinx}, die
enthaltenen Graphiken mittels
\href{http://grund-wissen.de/linux/tools/inkscape.html}{Inkscape} erstellt.
Der Quellcode sowie die Original-Graphiken  k\"{o}nnen \"{u}ber die
Projektseite heruntergeladen werden:
```

```
\textbf{http://www.grund-wissen.de}
\\[3pt]

Bei Fragen, Anmerkungen und Verbesserungsvorschl\"{a}gen bittet der Autor um
eine kurze Email an folgende Adresse:

\textbf{info@grund-wissen.de}
\\[5pt]

Augsburg, den \today. \\[6pt]
\@author

\setcounter{footnote}{0}%
  \let\thanks\relax\let\maketitle\relax
  %\gdef\@thanks{}\gdef\@author{}\gdef\@title{}
  \if@openright\cleardoublepage\else\clearpage\fi
  \spx@tempa
}
```

Zusätzlich habe ich in der Datei `writers/latex.py` beide Vorkommnisse der Bezeichnung „Release“ durch „Version“ ersetzt.

Linkcheck bei URLs mit Umlauten

Sphinx ist für englisch-sprachige Dokumentationen entworfen worden. Inzwischen kann beispielsweise eine deutsch-sprachige Sprache in der Konfigurationsdatei des jeweiligen Projekts mittels der Option `language=de` eingestellt werden; auch vielerlei weitere Sprachen werden inzwischen unterstützt.

Schwierigkeiten bereiten auch in der Sphinx-Version 1.4.5 immer noch Links auf Webseiten, die Umlaute in ihren URLs beinhalten; dies ist beispielsweise des öfteren bei Wikipedia-Seiten der Fall. Ruft man im Projektverzeichnis `make linkcheck` auf, so bricht der Test bei einer solchen URL mit einer Fehlermeldung ab.

Als kleinen Workaround habe ich in der Datei `builders/linkcheck.py` folgende Änderung an der Funktion `check_uri()` vorgenommen:

```
VORHER:

# handle non-ASCII URIs
try:
    req_url.encode('ascii')
except UnicodeError:

    (...)

NACHHER:

# handle non-ASCII URIs
try:
```

```
    req_url.encode('utf-8')
except UnicodeError:

    (...)
```

Trotz der obigen Änderung werden URLs mit Umlauten beim Linkcheck zwar als fehlerhaft angezeigt, der Test läuft jedoch durch. Dadurch kann der Linkcheck wie gewohnt genutzt werden, man muss lediglich bei URLs mit Umlauten die Fehler ignorieren oder diese manuell prüfen.

Das Blog-System Tinkerer

Tinkerer ist ein relativ junges Projekt, das auf *Sphinx* aufbaut und es erlaubt, ein statisches HTML-Weblog aus einer beziehungsweise mehreren Textdateien mit ReStructuredText-Syntax zu erzeugen. „Statisch“ soll hierbei bedeuten, dass es zwar keine Kommentier-Funktion und kein Content-Management-System bietet, dafür allerdings ohne jede Datenbank und serverseitige Skripte auskommt. Es genügt also bereits ein einfacher Webspace ohne Extras, um das Weblog zu hosten.

Tinkerer kann folgendermaßen installiert werden:

```
sudo pip3 install Tinkerer
```

Ruft man anscheinend in einer Shell `tinker` auf, so bekommt man eine kurze Hilfeseite mit den wichtigsten Programm-Optionen angezeigt.

Ein neues Weblog anlegen

Um mit Tinkerer ein Weblog zu generieren, legt man zunächst ein neues Verzeichnis an; anschließend wechselt man ins Projektverzeichnis:

```
# Blog-Verzeichnis anlegen:
mkdir myblog

cd myblog
```

Ähnlich wie Sphinx stellt auch Tinkerer eine Routine für das erstmalige Erstellen eines Blogs bereit. Diese wird folgendermaßen gestartet:

```
tinker --setup
```

Einige Basis-Einstellungen können anschließend in der neu erstellten Datei `conf.py` im Projektverzeichnis vorgenommen werden:

```
# Titel des Blogs festlegen:
project = 'Mein Blog'

# Zusatzzeile unter dem Blogtitel
```

```
tagline = 'Eine zusätzliche Header-Zeile'

# Kurze Beschreibung dse Blogs:
description = 'Sinn dieses Weblogs'

# Change this to your name
author = 'Vorname Nachname'

# Change this to your copyright string
copyright = '2016, ' + author

# Change this to your blog root URL (required for RSS feed)
website = 'http://www.grund-wissen.de/blog/'
```

Der angegebene Webseiten-Pfad ist wichtig, damit die Seite mit einem RSS-Reader abonniert werden kann und die Abonnenten so automatisch über neue Artikel informiert werden.

Gibt man in einer Shell `tinker` ohne weitere Angaben ein, so werden die möglichen Aufruf-Optionen eingeblendet.

Einen neuen Weblog-Eintrag erstellen

Um einen neuen Weblog-Eintrag („Posting“) zu erstellen, gibt man im Projektverzeichnis folgendes ein:

```
# Neuen Blog-Eintrag mit angegebenem Titel erstellen:
tinker -p 'Titel des Weblog-Eintrags'
```

Hierdurch wird im Projektverzeichnis eine neue RestructuredText-Datei beispielsweise unter `2016/03/23/titel_des_weblog_eintrags.rst` erstellt, wobei die Verzeichnisangabe dem aktuellen Datum entspricht. Diese Datei kann mit einem Texteditor geöffnet und mit Text gemäß der regulären RestructuredText-Syntax gefüllt werden.

Gleichzeitig wird in der Datei `master.rst` im Projektverzeichnis ein Eintrag ins dortige Inhalts-Verzeichnis aufgenommen:

```
# Datei master.rst

Sitemap
=======

.. toctree::
   :maxdepth: 1

   2016/03/23/titel_des_weblog_eintrags
```

Mag man beispielsweise nachträglich den Titel eines Eintrags ändern, so sollte sowohl der Name der `.rst`-Datei als auch die entsprechende Pfadangabe in der Datei `master.rst` angepasst werden.

Eine neue Weblog-Seite erstellen

Um eine neue, statische Weblog-Seite zu erstellen, gibt man im Projektverzeichnis folgendes ein:

```
# Neue Blog-Seite mit angegebenem Titel erstellen:
tinker --page 'Titel der Weblog-Seite'
```

Hierdurch wird im Projektverzeichnis eine neue RestructuredText-Datei `pages/titel_der_weblog_seite.rst` erstellt. Diese Datei kann wiederum mit einem Texteditor geöffnet und mit Text gemäß der regulären RestructuredText-Syntax gefüllt werden.

Auch in diesem Fall wird in der Datei `master.rst` im Projektverzeichnis ein Eintrag ins dortige Inhalts-Verzeichnis aufgenommen:

```
# Datei master.rst

Sitemap
=======

.. toctree::
   :maxdepth: 1

   2016/03/23/titel_des_weblog_eintrags
   pages/titel_der_weblog_seite
```

Soll also nachträglich der Name einer Seite geändert werden, so sollte wiederum sowohl der Name der `.rst`-Datei als auch die entsprechende Pfadangabe in der Datei `master.rst` angepasst werden.

Statische Seiten wie ein Impressum können beispielsweise verwendet werden, um Informationen über den Autor sowie Kontaktmöglichkeiten zu hinterlegen.

Weblog aus Quellcode-Dateien generieren

Um aus den `.rst`-Quelldateien fertige HTML-Seiten zu erzeugen, gibt man im Projektverzeichnis folgendes ein:

```
# Weblog aus Quellcode bauen:
tinker -b
```

Durch diesen Aufruf werden die fertigen HTMl-Dateien (bei Verwendung der Standard-Optionen) im Unterverzeichnis `blog` innerhalb des Projektverzeichnisses gespeichert. Von dort können sie mit einem FTP-fähigen Dateimanager, beispielsweise *mc*, auf den zur Domain gehörenden Webspace kopiert werden.

Links

Eine umfangreiche Dokumentation zu Sphinx mitsamt einem Schnell-Einstieg in Restructured Text und die Python-Matplotlib findet sich hier:

- Sphinx-Projektseite
- Sphinx Tutorial (en.)
- Sphinx Memo
- Matplotlib Documentation

Eine Liste alle hierfür verfügbarer `Roles` und `Directives` gibt es auf der Dokumentationseite von `Docutils` (Sphinx baut auf `Docutils` auf):

- Übersicht über Restructured-Text-Directives (en.)
- Übersicht über Restructured-Text-Roles (en.)

Um zu gewährleisten, dass die HTML-Ausgabe bzgl. der Syntax fehlerfrei ist, gibt es vom `W3C`-Consortium frei verfügbare Online-Prüfdienste:

- W3C (X)HTML Markup Validation Service
- W3C CSS Validation Service
- W3C Linkcheck

Versionskontrolle mit `git`

Das Versionssystem Git ist bei den meisten Distributionen bereits vorinstalliert. Möchte man Git mit einer graphischen Oberfläche benutzen, so können wahlweise giggle, gitk, qgit, git-gui oder git-cola als Pakete nachinstalliert werden; zudem existiert mit tig eine sehr konfigurierbare textbasierte Bedienoberfläche:

```
sudo aptitude install giggle gitk qgit git-gui git-cola tig
```

Die Bedienoberflächen können dann mittels `giggle` beziehungsweise `gitk`, `qgit`, `git gui`, `git-cola` beziehungsweise `tig` gestartet werden. Alle Programme können nach einer gewissen Einarbeitungszeit für ein schnelleres Arbeiten und für eine graphisch ansprechende Darstellung der Entwicklungszweige eines Projekts nützlich sein. Allerdings ist es auch bei Verwendung dieser Programme hilfreich, die wichtigsten Git-Anweisungen und Konzepte zu kennen.

Ausführliche englischsprachige Anleitungen zu den einzelnen Git-Anweisungen, die im folgenden vorgestellt werden, finden sich in den Linux-Manpages und können allgemein mit `man git-init`, `man git-config`, `man git-commit`, usw. aufgerufen werden.

Einführung: Ein neues Git-Projekt erstellen

Um ein neues Git-Projekt zu erstellen, kann man in einer Shell folgendes eingeben:

```
# Projekt initiieren:
mkdir git-test  && cd git-test
git init
```

Zunächst wird im obigen Beispiel ein neues Verzeichnis erstellt, anschließend wird in diesem ein neues Projekt initiiert. Dabei wird automatisch ein Unterverzeichnis `.git` angelegt; in diesem ist das gesamte Git-Repository inklusive der kompletten Versionsgeschichte in komprimierter Form gespeichert.

Konfiguration von Git

Jedes `.git`-Verzeichnis enthält unter anderem eine Konfigurationsdatei namens `config`, die standardmäßig folgenden Inhalt hat:

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
```

Die Datei sollte bei einer erstmaligen Verwendung von Git um folgenden Eintrag ergänzt werden:

```
[user]
    name="Vorname Nachname"
    email="email@adresse.de"
```

Möchte man Git für eigene Projekte häufiger verwenden, ist eine „globale“ Konfiguration sinnvoll, die nicht nur für das aktuelle Projekt, sondern für alle Projekte des Benutzers gilt. Derartige Einstellungen werden in der Datei `~/.gitconfig` gespeichert. Die Konfigurationen können entweder mittels eines Texteditors direkt in dieser Datei vorgenommen werden, oder in entsprechender Weise mittels `git config` gesetzt werden:

```
# Globale Variablen festlegen:
git config --global user.name "Vorname Nachname"
git config --global user.email "email@adresse.de"
```

Mit `git config -l` können die bestehende Einstellungen angezeigt werden.

Veränderungen speichern

Nimmt man innerhalb des Projektverzeichnisses eine Änderung vor, erstellt beispielsweise mittels `touch README` eine leere Datei, so kann man anschließend mittels `git add README` die neue Datei, oder einfacher `git add *` alle neuen Dateien im Verzeichnis in den Index der zu berücksichtigenden Dateien hinzufügen:

```
# Dateien in Versions-Index aufnehmen:
git add *
```

Bei Benutzung von `git add` kann ein beliebiges Dateimuster angegeben werden, beispielsweise würden durch `git add *.txt` alle Dateien mit der Endung `.txt` in den Index der zu versionierenden Dateien aufgenommen.[1]

Die Änderungen können anschließend mittels `git commit` gespeichert werden:

```
# Aktuelle Version der Dateien im Index speichern:
git commit -m "Initial Commit: Adding a README file."
```

Die Option `-m "Beschreibung"` fügt eine obligatorische Nachricht dem hinzu, die in Kurzform den Grund des Commits beschreibt; Lässt man diese Option weg, so wird durch die `commit`-Anweisung automatisch der durch die Shell-Variable `$EDITOR` festgelegte Standard-Texteditor zur Eingabe einer Commit-Nachricht geöffnet.

Git speichert bei einem Commit nur diejenigen Dateien, die mittels `git add` in den Index aufgenommen und seitdem nicht verändert wurden; sollen auch seit dem letzten Commit geänderte Dateien gespeichert werden, so müssen sie erneut mittels `git add` in den Index des nächsten Commits hinzugefügt werden. Eine wesentlich praktischere Möglichkeit besteht in der Definition von folgendem Alias in der Konfigurationsdatei `~/.gitconfig`:

```
[alias]
    addremove = !git add . && git ls-files --deleted | xargs git rm
```

Mit dieser Definition kann vor jedem Commit `git addremove` eingegeben werden, um alle neu hinzugekommenen und veränderten Dateien in den Index aufzunehmen:

```
# Alle Änderungen im Index berücksichtigen:
git addremove
```

Ein entscheidender Vorteil hiervon besteht darin, dass dadurch auch gelöschte Dateien aus dem Index entfernt sowie lediglich umbenannte Dateien als solche erkannt werden; dies kann mit dazu beitragen, das Repository so klein wie möglich zu halten.[2]

Status und Veränderungen anzeigen

Den bisherigen Versionsverlauf kann man sich mit `git log` anzeigen lassen. Dieser sieht nach dem ersten Commit etwa wie folgt aus:[3]

[1] Die Dateien, die im Index eines Git-Repositories vermerkt sind, können folgendermaßen aufgelistet werden:

```
git ls-files --abbrev --stage
```

Durch die zusätzliche Option `--abbrev` werden nur die ersten sieben Zeichen der Datei-IDs anstelle der kompletten IDs angezeigt.

[2] Ebenfalls zum „guten Stil“ gehört es, in Textdateien konsequent auf unnötige Leerzeichen am Zeilenende zu achten; auch ein Hinzufügen oder Entfernen von Leerzeichen wird bei neuen Commits als Änderung der jeweiligen Datei angesehen und bewirkt somit unnötige Einträge in die Versionsgeschichte.

[3] Die Log-Nachrichten können in einer übersichtlicheren Version ausgegeben werden (einzeilig, mit abgekürzten Commit-Hashes und Syntax-Highlighting), indem man sich folgende Abkürzung („Alias“) definiert:

```
git log

# Ergebnis:

# commit 9812b1b0121ac9159e745ba87a0cd31c9306e3bc
# Author: Bernhard Grotz <info@grund-wissen.de>
# Date:   Sun Mar 29 10:43:03 2015 +0200

# Initial commit: Adding a README file.
```

Auch für Anzeigen der Geschichte eines Git-Repositories können Alias-Definitionen in der `~/.gitcoinfig` hilfreich sein, beispielsweise:

```
[alias]
    l = log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s \
        %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative
```

Gibt man mit der obigen Definition `git l` ein, so erhält man eine farbig hervorgehobene Anzeige der Versionsgeschichte.

Jeder Commit wird in Git durch eine 40 Zeichen lange Hash-ID-Zeichenkette symbolisiert. Möchte man einen bestimmten Commit besonders hervorheben, um beispielsweise auf eine neue Funktion in einem Software-Projekt hinzuweisen, so kann man den Commit mit einem „Tag“ versehen:

```
git tag -a tagname -m "Add a comment here."
```

Um anzuzeigen, inwiefern sich der aktuelle Stand des Projekts vom Stand des letzten Commits unterscheidet, kann `git status` aufgerufen werden. Wurde noch keine weitere Veränderung vorgenommen, so zeigt diese Anweisung folgendes an:

```
git status

# Ergebnis:

# Auf Branch master
# nichts zu committen, Arbeitsverzeichnis unverändert
```

Ändert man die README-Datei etwas ab, beispielsweise mittels `echo "Hallo Git!" > README`, so zeigt `git status` folgendes an:

```
git status

# Ergebnis:

# Auf Branch master
```

`git config --global alias.lg "log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative"`

Nach dieser Definition wird in allen Repositories des Benutzers `git lg` als neue Abkürzung erkannt und dabei. (Dieser Hinweis stammt ursprünglich von Filipe Kiss).

```
# Änderungen, die nicht zum Commit vorgemerkt sind:
#   (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit␣
↪vorzumerken)
#   (benutzen Sie "git checkout -- <Datei>...", um die Änderungen im␣
↪Arbeitsverzeichnis zu verwerfen)

#     geändert:               README

# keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git␣
↪commit -a")
```

Um die Änderungen in die Versionierung zu übernehmen, kann man wiederum `git commit -am "Beschreibung"` aufrufen.

Dateien ignorieren

Bei Verwendung von Git gibt es zwei Möglichkeiten, Dateien von der Versionierung auszunehmen:

- In einer Datei `.gitignore` im Projektverzeichnis werden Dateien oder Dateimuster festgelegt, die innerhalb des lokalen Verzeichnisses von Git ignoriert werden sollen.
- In der Datei `~/.git/info/exclude` können Dateien oder Dateimuster angegeben werden, die unabhängig von einem konkreten Projekt stets von Git ignoriert werden sollen.[4]

In einer Ignore-Datei können die in der Shell üblichen Dateimuster genutzt werden:

```
# Alle Dateien in "_build"-Verzeichnis ignorieren:
_build/

# Dateimuster ignorieren:
*.pyc
*.o
*.aux
*.swp
*.log
*.tmp
```

Sollen Dateien oder Verzeichnisse nur dann ignoriert werden, wenn sich diese unmittelbar im Projektverzeichnis, aber nicht in einem Unterverzeichnis befinden, so kann vor den Datei- beziehungsweise Verzeichnisnamen ein / vorangestellt werden. Git interpretiert dies als Zeichen für die erste Ebene des Projektverzeichnisses, nicht wie die Shell als Quelle des Verzeichnisbaums.

[4] Es kann anstelle von `~/.git/info/exclude` auch eine andere Datei als „globale“ Ignore-File festgelegt werden. Die Syntax hierfür lautet beispielsweise `git config --global core.excludesfile ~/.gitignore`.

Soll eine Datei oder ein Verzeichnis explizit beachtet werden, obwohl es auf ein Ignore-Muster zutrifft, kann unmittelbar vor das Muster (ohne Leerzeichen dazwischen) ein `!` geschrieben werden.

Ausführliche Beschreibungen zu `.gitignore`-Dateien und Shell-Dateimustern kann man in den Manualseiten mittels `man gitignore` beziehungsweise `man glob` oder unter https://git-scm.com/docs/gitignore nachlesen.

Arbeitsverzeichnis, Index und Objektspeicher

In Git wird ein Projektverzeichnis mitsamt allen Versionen der verwalteten Dateien als Repository bezeichnet. In jedem solchen Repository gibt es drei verschiedene Speicher-Ebenen:

- Als Arbeitsverzeichnis („working directory“) wird das Projektverzeichnis in der aktuellen Version bezeichnet; Dateien früherer Versionen sind darin nicht unmittelbar sichtbar.
- Als Index („stage“) wird die Zwischenebene bezeichnet, die beim nächsten Aufruf von `git commit` die nächste Instanz des Arbeitsverzeichnisses ausmacht. Dateien, die sich im Index befinden, werden als „staged“ bezeichnet. Veränderte Dateien werden allgemein mittels `git add` in den Index übernommen.
- Als Objektspeicher bezeichnet man die Datenbank, in welcher auch die vergangenen Versionen des Repositorys gespeichert sind. Durch `git commit` werden die Änderungen aus dem Index in den Objektspeicher übernommen.

Der „Lebenszyklus“ von versionierten Dateien

Dateien, die in einem Projektverzeichnis neu erstellt werden, werden von Git nicht automatisch in die Versionierung aufgenommen – sie sind „unstaged“ und müssen erst mittels `git add` explizit in den Index aufgenommen werden. Anschließend werden so hinzugefügte Dateien als „unmodified“ angesehen. Diesen Status haben ebenso alle Dateien, die seit dem letzten Commit nicht verändert wurden.

Werden bestehende, von Git berücksichtigte Dateien verändert, so ändert sich ihr Status in „modified“. Mittels `git add` können sie zum Index hinzugefügt werden. Nach einem Commit sind all diese Dateien (in der neu gespeicherten Version) wiederum „unmodified“.

Sollen die Änderungen einer Datei beim Commiten unberücksichtigt bleiben, so kann die Datei mittels `git rm --cached dateiname` wieder aus dem Index gelöscht werden; die Datei bleibt dann als „unstaged“ im Arbeitsverzeichnis bestehen. Wird die Option `--cached` weggelassen, wird die Datei sowohl aus dem Index wie auch aus dem Arbeitsverzeichnis gelöscht.

Arbeiten mit Entwicklungszweigen (Branching)

Nach dem ersten Commit wird von Git automatisch ein Entwicklungszweig („Branch“) namens `master` eingerichtet. Möchte man nun am bestehenden Projekt experimentieren,

beispielsweise neue Funktionen ausprobieren, so kann man dies mittels eines neuen Entwicklungszweigs tun, ohne dass dies Auswirkungen auf den eigentlichen `master`-Branch hat.

Ein neuer Branch wird mittels `git branch` angelegt:

```
# Entwicklungszweig des Projekts erstellen:
git branch dev
```

Wird `git branch` ohne weitere Argumente aufgerufen, so werden alle Branches des Projekts aufgelistet, wobei der aktuell ausgewählte Entwicklungszweig mit einem `*`-Zeichen markiert ist.

```
# Branches anzeigen:
git branch

# Ergebnis:

    dev
  * master
```

Ein Wechsel zwischen den einzelnen Entwicklungszweigen ist mittels `git checkout` möglich:

```
# In den dev-Branch wechseln:
git checkout dev
```

Git erlaubt nur dann einen Checkout eines anderen Branches, wenn der aktuelle Branch „clean“ ist, also keine Änderungen zum Committen anstehen. Möchte man dennoch ohne neuen Commit einen Branch verlassen, so können die in der Zwischenzeit vorgenommenen Änderungen mittels `git stash` zwischengespeichert werden. Mittels `git stash list` kann angezeigt werden, ob aktuell in einem Entwicklungszweig Änderungen zwischengespeichert wurden; im jeweiligen Entwicklungszweig können mittels `git stash apply` die zwischengespeicherten Änderungen wiederum übernommen werden.

Im Projektverzeichnis ist immer nur ein einzelner Branch „aktiv“. Hat man beispielsweise den Branch `dev` ausgewählt und führt dort einen Commit durch, so ist dieser Commit nur für diesen Branch wirksam. Wenn dann bei einer neueren Version im `dev`-Branch Dateien im Projektverzeichnis erstellt zur Versionierung hinzugefügt wurden, so werden diese ebenfalls nur dann im Arbeitsverzeichnis angezeigt, wenn der zugehörigen Entwicklungsbranch aktiv ist. Git speichert die Dateien intern im Objektspeicher, löscht sie gegebenenfalls beim Verlassen den Entwicklungsbranches und fügt sie automatisch wieder ins Arbeitsverzeichnis ein, wenn der Entwicklungsbranch wieder aktiviert wird.[5]

Branches kann man sich allgemein als Zeiger auf einzelne Commits vorstellen. Sie helfen dabei, ein Projekt in logische Teile zu untergliedern. Man sollte allgemein *früh* und *oft* neue Branches bei der Entwicklung eines Projekts anlegen.

[5] Git speichert beim Commits und Branches nicht den gesamten Inhalt aller Dateien, sondern (in komprimierter Form) nur die jeweiligen Änderungen relativ zur vorhergehenden Version.

Zusammenführen von Entwicklungszweigen (Merging)

Um die Entwicklungen eines Branches in einen anderen Branch zu übernehmen, wird zunächst mittels `git checkout` der Zielbranch ausgewählt. Von diesem aus wird dann `git merge` unter Angabe des einzubindenden Entwicklungsbranches aufgerufen.

Angenommen, im `master`-Branch befindet sich eine Datei `file.txt`. Nach dem Erstellen und Auswählen eines entsprechenden Branches soll die Datei im neuen Entwicklungszweig geändert werden:

```
# Branch erstellen und eine Datei ändern:
git branch test
git checkout test
echo "Test Test Test" > file.txt
```

Sollten die Datei `file.txt` oder andere Dateien im Entwicklungszweig beim Aufruf von `git status` als „untracked“ angezeigt werden, so kann mittels `git addremove` der Index komplett aktualisiert werden – neue Dateien Dateien werden dabei zum Index hinzugefügt, manuell gelöschte Dateien entfernt.

Nach einem `git commit` im neuen `test`-Branch eilt dieser dem `master`-Branch in der Entwicklung voraus, wie man mittels `git log` erkennen kann. Sollen die Änderungen in den `master`-Branch übernommen werden, gibt man folgendes ein:

```
# master-Branch auswählen und Änderungen übernehmen:
git checkout master
git merge test
```

Sofern sämtliche Entwicklungen im Entwicklungsbranch – wie im obigen Beispiel – aktueller sind als im Zielbranch, funktioniert ein Merge stets ohne Probleme: Die Änderungen werden im Zielbranch übernommen und die Version des Zielbranches automatisch angepasst.

Konkurrierende Änderungen verwalten

Konflikte können auftreten, wenn nach dem Erstellen eines neuen Branches Veränderungen sowohl im `master`- wie auch im Entwicklungsbranch vorgenommen werden:

```
# Änderung im test-Branch vornehmen:
git checkout test
echo "La La La" >> file.txt

git commit -am "changing file.txt"

# Änderung im master-Branch vornehmen:
git checkout master
echo "Ha Ha Ha" >> file.txt

git commit -am "changing file.txt"
```

Versucht man nun im `master`-Branch mittels `git merge` die Änderungen aus dem Entwicklungsbranch zu übernehmen, so bekommt man eine Fehlermeldung angezeigt, da Git nicht weiß, in welche Richtung die Änderungen übernommen werden sollen:

```
git merge test

# Ergebnis:
# Automatisches Zusammenfügen von file.txt
# KONFLIKT (Inhalt): Merge-Konflikt in file.txt
# Automatischer Merge fehlgeschlagen; beheben Sie die Konflikte und committen␣
↪Sie dann das Ergebnis.
```

Eine einfache Methode, um einen solchen aus konkurrierenden Änderungen resultierenden Konflikt zu beheben, ist der Aufruf von `git mergetool`. Ist kein Standard-Werkzeug durch eine Option vorgegeben (beispielsweise mittels `git config --global merge.tool vimdiff`), so kann ein zur Verfügung stehendes Werkzeug zum Anzeigen der konkurrierenden Änderungen ausgewählt werden.

Die zu ändernden Stellen werden von Git folgendermaßen gekennzeichnet:

```
<<<<<<< HEAD:file.txt
Diese Änderungen wurden im master-Branch vorgenommen
=======
Diese Änderungen wurden im test-Branch vorgenommen
>>>>>>> test:file.txt
```

Um das Merging abzuschließen, muss manuell eine der konkurrierenden Stellen ausgewählt oder eine andere Änderung vorgenommen werden; die Marker müssen dabei ebenfalls entfernt werden, da erst dann die Konflikte von Git als bereinigt angesehen werden.

Anschließend können die Änderungen mit `git commit -am "Beschreibung"` dem Index hinzugefügt und gespeichert werden.

Arbeiten mit externen Repositories

Git als Versionskontrollsystem wurde vorrangig entwickelt, um die Zusammenarbeit zwischen mehreren Entwicklern zu erleichtern. Ein Gedanke dabei war und ist, dass ein gemeinschaftlich bearbeitetes Projekt auf einem zentralen Server liegt, die einzelnen Entwickler sich Kopien dieses Projekts herunterladen können, lokal Entwicklungen vornehmen und diese schließlich wieder in das zentral gespeicherte Projekt einfließen lassen.

Externe Repositories herunterladen

Um ein existierendes Repository von einem externen Server, beispielsweise von GitHub oder Bitbucket herunterzuladen, gibt man in der Shell folgende Anweisung an:

```
# Allgemein: git clone https://github.com/UserName/RepositoryName.git
```

```
# Beispiel:
 git clone https://github.com//grund-wissen/grundkurs-linux.git
```

Die `clone`-Anweisung bewirkt, dass eine vollständige Kopie des Repositorys (mitsamt Versionsgeschichte) heruntergeladen und als neuer Unterordner im aktuellen Verzeichnis gespeichert wird. Mit dieser Kopie des Repositorys kann lokal wie in jedem anderen Repository gearbeitet werden.

Eigene Repositories hochladen

Die am meisten verwendete Methode, um Repositories anderen Entwicklern auf einem zentralen Server zugänglich zu machen, ist die Nutzung von GitHub oder Bitbucket. Auf beiden Webseiten wird kostenloser Speicherplatz für öffentliche Repositories angeboten (private Repositories sind oftmals kostenpflichtig). Auf beiden Webseiten muss entsprechend der Angaben auf der jeweiligen Webseite ein Nutzer-Account unter Angabe eines Benutzernamens, eines Passworts und einer Emailadresse erstellt werden.

Innerhalb des Nutzer-Accounts von beispielsweise GitHub wird mittels der Web-Oberfläche ein neues Repository angelegt. Es ist empfehlenswert, aber nicht zwingend notwendig, dieses ebenso zu nennen wie das lokale Projektverzeichnis. Das auf diese Weise neu angelegte Repository wird dann folgendermaßen als Quelle („origin“) des lokalen Repositorys festgelegt:

```
# Allgemein: git add origin https://github.com/UserName/RepositoryName.git

# Beispiel:
git remote add origin https://github.com/grund-wissen/grundkurs-linux.git
```

Ist das lokale Repository ein Clon eines externen Repositorys, so ist die `origin`-Variable bereits gesetzt. Mit `git remote -v` werden die entsprechenden Adressen und Branches angezeigt.

Die zu einem Remote-Namen gehörende Adresse kann bei Bedarf folgendermaßen geändert werden:

```
git remote set-url origin https://github.com/new-name
```

Bevor das lokale Repository hochgeladen wird, sollte noch eine Datei `README.rst` oder `README.md` (wahlweise mit ReStructuredText oder MarkDown -Syntax) für eine kurze Projektbeschreibung sowie eine `LICENSE`-Datei mit Lizenz-Hinweisen hinzugefügt werden. Auch eine `AUTHORS`-Datei mit einer Auflistung der aktiv am Projekt beteiligten Entwickler ist durchaus üblich.

Das Hochladen funktioniert mittels der Anweisung `git push`:

```
git push -u origin master
```

Die `push`-Anweisung bewirkt, dass alle lokal neu hinzugekommenen Commits in das externe Repository übernommen werden. Die Option `-u` bewirkt eine Synchronisierung beider

Repositories und sollte immer dann verwendet werden, wenn möglicherweise mehrere Entwickler am gleichen Branch arbeiten.

Lokale Repositories aktualisieren

Um neue Commits von einem externen Reposotory in ein zugehöriges lokales Repository zu übernehmen, können die Anweisungen `git pull` oder `git fetch` verwendet werden:

- `git pull` holt die Änderungen aus dem externen Repository und fügt diese dem Index der beim nächsten Commit zu berücksichtigenden Dateien hinzu.
- `git fetch` führt zunächst `git pull` aus, gefolgt von `git merge`.

Voraussetzung für beide Anweisungen ist wiederum (wie im letzten Abschnitt beschrieben), dass ein externes Repository als `origin` festgelegt ist. Gibt es konkurrierende Änderungen, so müssen diese wiederum, wie im Abschnitt *Merging* beschrieben, manuell beispielsweise mit `git mergetool` in Einklang gebracht werden.

Forks und Pull Requests

GitHub bietet unter der Bezeichnung „Fork“ eine komfortable Methode, Repositories von einem anderen Entwickler als Clone in den eigenen Benutzeraccount zu übernehmen. Dazu genügt ein Klick auf den Fork-Button im entsprechenden Repository.

An diesem „Fork“ des Original-Projekts kann man nun arbeiten und entwickeln werden, ohne Angst haben zu müssen, im Original etwas kaputt machen zu können. Hat man einen Teil des Codes ergänzt oder verbessert, so kann man durch einen Klick auf den entsprechenden Button im geforkten Repository dem Maintainer des Originals einen „Pull Request“ senden, durch den dieser eine Benachrichtigung über die neue Entwicklung erhält. Der Maintainer kann dann entscheiden, ob er diese Änderungen des Codes übernimmt oder nicht.

Bei der Arbeit mit der lokalen Kopie des Repositories gibt es nun allerdings *zwei* externe Repositories, mit denen der Clon in Kontakt steht: Das Haupt-Repository und den Fork. Praktischerweise richtet man sich dafür eine zweite Remote-Adresse ein, die man beispielsweise „upstream“ nennt, weil sie die Hauptquelle angibt; „origin“ sollte dann entsprechend auf den Fork zeigen.

```
git remote add upstream https://github.com/path-to-main-repository
git remote set-url origin https://github.com/path-to-fork-repository
```

Mit diesen Einstellungen können nun neue Commits aus dem `upstream`-Repository geholt und mit den lokalen Änderungen gemerged werden; das Ergebnis kann wiederum in das `origin`-Repository gepushed werden.

... to be continued ...

Links

- Git Tutorial 1
- Git Tutorial 2
- Git Tutorial 3
- Git Tutorial 4
- Git Tutorial 5
- Learning Git Branching

Der Texteditor `vim`

Vim ist einer der vielseitigsten Text-Editoren aller Zeiten. Man kann damit auf sehr effiziente Weise nicht nur Programme schreiben, sondern auch Webseiten, Bücher, Briefe, Emails, kurz: Textdateien aller Art bearbeiten.

Durch Tastenkombinationen (insbesondere durch *Snippets*) kann die die zum Schreiben von Texten nötige Tipparbeit ist erheblich reduziert werden; nach einer gewissen Einarbeitungszeit wird das Arbeitstempo so spürbar erhöht. Bei häufiger Benutzung laufen einzelne Editierungsschritte mitunter so schnell und „automatisch“ ab, dass man nur noch unterbewusst darüber nachdenkt und die Konzentration der eigentlichen Arbeit widmen kann.. ;-)

Installation

Auf fast allen Linux-System ist `vi`, der „kleine Bruder“ von Vim, bereits in der Grundversion enthalten. Für eine komfortablere Bedienung ist es allerdings empfehlenswert, eine umfangreichere Version des Vim zu installieren:[1]

```
sudo aptitude install vim-common vim-gnome vim-scripts vimhelp-de ncurses-term
```

Mit den obigen Paketen wird die Basis-Version von Vim installiert, die in einer Shell mittels `vim` gestartet werden kann. Zudem wird eine graphische Bedienoberfläche namens `gvim` installiert, die als eigenständiges Programm ohne Shell-Fenster aufgerufen werden kann.

Vim-Schnellstart

Bei der im letzten Abschnitt beschriebenen Installation wird automatisch das Vim-Lern-Programm `vimtutor` mit installiert, das in einer Shell gestartet werden kann:

[1] Vim läuft auch unter Windows- und Apple-Systemen. Hierfür finden sich auf der Vim-Projektseite unter der Rubrik „Downloads“ entsprechende Pakete bzw. selbst entpackende Installationsdateien.

```
vimtutor
```

Hierbei wird Vim in einer Basis-Version gestartet und eine Tutorial-Datei aufgerufen, in der interaktiv in mehreren Lektionen die wichtigsten Vim-Tasten erlernt werden können. Für das Durchlaufen dieses Tutorials sollte man etwa 30 Minuten einplanen.

Bedienung

Vim wird meist komplett über die Tastatur gesteuert. Damit es bei einer begrenzten Anzahl an Tasten möglich ist schnell durch Dateien zu navigieren, Text einzugeben sowie Editierbefehle auszuführen, wurden verschiedene Ebenen („Modi“) eingeführt. Je nach aktivem Modus haben die Tasten verschiedene Bedeutungen.

- Beim Start von Vim befindet man sich im Normalmodus.
- In einen anderen Modus gelangt man durch Drücken einer entsprechenden Taste.
- Durch Drücken von `ESC` gelangt man in den Normalmodus zurück.

Um Vim zu beenden, kann man im Normalmodus `ZQ` (schließen, Änderungen verwerfen) beziehungsweise `ZZ` (schließen, Änderungen speichern) eingeben.

Einfügemodus

Der Einfügemodus dient – wie der Name schon sagt – zum Einfügen von Text in die aktuelle Datei. Hier gleicht Vim einem „normalen“ Editor: Alle Buchstaben, Zahlen, etc. erscheinen nach der Eingabe an der aktuellen Position auf dem Bildschirm.

Ausgehend vom Normalmodus gelangt man in den Einfügemodus durch Drücken einer der folgenden Tasten:

`i`	Text einfügen vor der momentanen Position	(*insert*)
`I`	Text einfügen zu Beginn der momentanen Zeile	
`a`	Text einfügen nach der momentanen Position	(*append*)
`A`	Text einfügen am Ende der momentanen Zeile	
`o`	Text einfügen unter der aktuellen Zeile	(*open a new line*)
`O`	Text einfügen über der aktuellen Zeile	

Auch mit `s`, `S`, `c`, `C`, `r`, `R` gelangt man in den Einfügemodus. Hierbei wird jedoch bestehender Text ersetzt.

Gibt man beispielsweise `10i` gefolgt von etwas Text ein, so wird nach Beenden der Eingabe mit `Esc` das 10-fache des eingegebenen Textes eingefügt.

Im Einfügemodus gibt es standardmäßig folgende Tastenkombinationen:

`Ctrl n` bzw. `Ctrl p`	Vervollständigung des aktuellen Wortes anhand eines bereits vorkommenden Wortes	(*next* bzw. *previous word*)
`Ctrl x` `Ctrl l`	Vervollständigung der aktuellen Zeile anhand einer bereits vorkommenden Zeile	(*line-completion*)
`Ctrl x` `Ctrl f`	Vervollständigung der Eingabe von Dateinamen und Pfaden	(*file-completion*)

Snippets

Durch die Verwendung von so genannten „Snippets“ kann viel Tipparbeit gespart werden. Das Prinzip hierbei ist sehr einfach: Man definiert sich frei wählbare Abkürzungen, die dann nach einem Drücken der `Tab`-Taste (oder einer anderen frei wählbaren Taste) ein ganzes Template ausrollen können, so dass nur noch an einzelnen Stellen Text eingefügt werden muss.

Das meiner Meinung nach beste Snippet-Plugin ist *Ultisnips*; man kann damit je nach Dateityp beziehungsweise Endung einer Datei unterschiedliche Abkürzungen definieren. Die Abkürzungen liegen allesamt in einem gemeinsamen Ordner und sind jeweils mit `dateityp.snippets` benannt. Beispielsweise kann man sich so sowohl für C-Dateien als auch für Python-Dateien je ein Snippet mit der Abkürzung `ife` definieren, das dann nach einem Drücken der `Tab`-Taste im jeweiligen Dateityp eine `if-else`-Abfrage mit Sprungmarken zu den jeweiligen Eingabestellen erstellt.

Besonders praktisch sind Snippets für ein Arbeiten mit LaTeX -Dateien: So kann man beispielsweise ein Snippet namens `art` definieren, das eine gesamte Standard-Präambel für die Dokumentklasse `article` definiert. So genügt es dann, eine leere `.tex`-Datei zu öffnen, im Einfügemodus `art` und `Tab` einzugeben, und man kann bereits mit dem Schreiben des eigentlichen Dokuments beginnen. :-)

Snippet-Beispieldateien sind auf der Grund-Wissen-Seite verlinkt.

Normalmodus

Der Normalmodus dient der Navigation durch die geöffnete(n) Datei(en) und der Bearbeitung von deren Inhalt. Jede Taste besitzt hier ihre eigene Funktion. (Ein Spickzettel kann in der Lernphase hilfreich sein!)

Eigene Zuweisungen von beliebigen Funktionen auf Tasten(kombinationen) lassen sich mittels *Mappings* definieren.

Bewegen

Die folgenden Tasten(kombinationen) bewirken im Normalmodus eine Bewegung des Cursors.

`hjkl` kann alternativ zu den Pfeiltasten genutzt werden:

`j`	↓	Gehe eine Zeile nach unten
`k`	↑	Gehe eine Zeile nach oben
`h`	←	Gehe eine Stelle nach links
`l`	→	Gehe eine Stelle nach rechts

`Ctrl f` und `Ctrl b` entsprechen `PageDOWN` und `PageUp`:

`Ctrl f` oder `PageDOWN`	Gehe eine Seite nach unten
`Ctrl b` oder `PageUP`	Gehe eine Seite nach oben

Um zu einer bestimmten Zeile zu gelangen, gibt es folgende Tastenkürzel:

`gg`	Gehe zum Anfang der Datei
`G`	Gehe ans Ende der Datei

Mit der Eingabe von `nG` oder `ngg` springt man zur `n`-ten Zeile. Beispielsweise gelangt man durch die Eingabe von `3gg` beziehungsweise `3G` zur dritten Zeile.

Navigation innerhalb einer Zeile

Mit `0` und `$` gelangt man schnell an den Anfang oder das Ende der aktuellen Zeile. Einzelne Buchstaben lassen sich gezielt mit `f` und `t` ansteuern:[1]

`^` oder `0`	Gehe an den Anfang der Zeile
`$`	Gehe an das Ende der Zeile
`f Buchstabe`	Gehe zu dem nächsten Vorkommen von `Buchstabe` (exakt)
`F Buchstabe`	Gehe zu dem vorherigen Vorkommen von `Buchstabe` (exakt)
`t Buchstabe`	Gehe zu dem nächsten Vorkommen von `Buchstabe` (eine Stelle vorher)
`T Buchstabe`	Gehe zu dem vorherigen Vorkommen von `Buchstabe` (eine Stelle vorher)
`;` (Strichpunkt)	Wiederhole die letzte Buchstabensuche in gleicher Richtung
`,` (Komma)	Wiederhole die letzte Buchstabensuche in umgekehrter Richtung

Um von einem Wort zum nächsten zu gelangen, gibt es folgende Tastenkürzel:

[1] Bei der aktuellen Version von Vim habe ich feststellen müssen, dass das `,`-Mapping mit standardmäßig mitinstallierten Plugin „justify.vim“ kollidiert; man muss, um dies zu beheben, mit SuperUser-Rechten die Datei `/usr/share/vim/vim74/pack/dist/opt/justify/plugin/justify.vim` editieren und folgende beiden Zeilen durch ein `"`-Zeichen am Zeilanfang auskommentieren:

```
" nmap ,gq :%s/\s\+/ /g<CR>gq1G
" vmap ,gq :s/\s\+/ /g<CR>gvgq
```

`w`	Gehe an den Anfang des nächsten Wortes	`W`	wie `w`, jedoch ohne Rücksicht auf Satzzeichen
`e`	Gehe an das Ende des aktuellen Wortes	`E`	wie `e`, jedoch ohne Rücksicht auf Satzzeichen
`b`	Gehe an den Anfang des vorherigen Wortes	`B`	wie `b`, jedoch ohne Rücksicht auf Satzzeichen
`ge`	Gehe an das Ende des vorherigen Wortes		

Navigation zwischen Sätzen und Abschnitten

Für Sprünge zum nächsten beziehungsweise vorherigen Satz oder Abschnitt lassen sich die Klammer-Symbole verwenden:

`(`	Gehe an den Anfang des aktuellen Satzes	`)`	Gehe an den Anfang des nächsten Satzes
`{`	Gehe zu der vorherigen leeren Zeile	`}`	Gehe zu der nächsten leeren Zeile
`[[`	Gehe zu der vorherigen Überschrift	`]]`	Gehe zu der nächsten Überschrift

Innerhalb des aktuellen Bildschirms kann man sich wie folgt bewegen:

`H`	Gehe zu der obersten Zeile des Bildschirms
`M`	Gehe zu der mittleren Zeile des Bildschirms
`L`	Gehe zu der untersten Zeile des Bildschirms

Hierbei gelangt man mit z.B. `5H` zur fünften Zeile des aktuellen Bildschirms, mit `2L` zur vorletzten.

Manchmal (z.B. beim Aufnehmen von *Makros*) finden auch folgende Tasten Verwendung:

`+`	Gehe zum ersten Zeichen der folgenden Zeile
`-`	Gehe zum ersten Zeichen der vorherigen Zeile
`n|`	Gehe zum `n`-ten Zeichen der aktuellen Zeile
`gm`	Gehe zur Fenstermitte (horizontal)
`%`	Gehe zum passenden Gegenstück einer Klammer
`Ctrl d`	Gehe eine halbe Seite nach unten
`Ctrl u`	Gehe eine halbe Seite nach oben
`Ctrl e`	Scrolle den Bildschirm eine Zeile nach oben (der Cursor bleibt unverändert)
`Ctrl y`	Scrolle den Bildschirm eine Zeile nach unten (der Cursor bleibt unverändert)
`zt`	Scrolle den Cursor an das obere Ende des Bildschirms
`zb`	Scrolle den Cursor an das untere Ende des Bildschirms
`zz`	Scrolle den Cursor in die Mitte des Bildschirms

Weitere Navigationsmöglichkeiten

Folgende Tastenkombinationen sind im Normalmodus ebenfalls oftmals hilfreich:

`gf`	Öffne die Datei unter dem Cursor
`gx`	Öffne den Link unter dem Cursor im Standard-Webbrowser (beispielsweise Firefox)

Beide Tastenkombinationen funktionieren nur, wenn sich der Cursor aktuell über einem existierenden Dateipfad oder einer Web-Adresse befindet.

Suchen

Im Normalmodus kann man mit `/` beziehungsweise `?` die aktuelle Datei vorwärts beziehungsweise rückwärts nach Textstellen durchsuchen:

`/ Suchbegriff`	Gehe zu dem nächsten Vorkommen von `Suchbegriff`
`? Suchbegriff`	Gehe zu dem vorherigen Vorkommen von `Suchbegriff`
`*`	Gehe zu dem nächsten Vorkommen des Worts unter dem Cursor
`#`	Gehe zu dem vorherigen Vorkommen des Worts unter dem Cursor
`gd`	Gehe zu dem ersten Vorkommen des Worts unter dem Cursor (*go [to the] definition*)

Ist die Option `hlsearch` in der *Konfigurationsdatei* gesetzt, so werden die Treffer farblich hervorgehoben. Mittels `:nohl` („no-highlight“) oder einem entsprechenden *Mapping* kann die Hervorhebung wieder aufgehoben werden.

Zu dem jeweils nächsten Treffer gelangt man mit `n`:

`n`	Gehe zum nächsten Treffer
`N`	Gehe zum nächsten Treffer (umgekehrte Richtung)

Mit `gD` kann auch nach einer globalen Definition (in allen geöffneten Buffern gesucht werden.

Bearbeiten

Im Normalmodus gibt es folgende Anweisungen, um Text zu kopieren, löschen, abzuändern, oder einzufügen:

`y`	Kopieren	(*yank*)
`d`	Löschen bwz. Ausschneiden	(*delete*)
`c`	Ändern	(*change*)
`p`	Einfügen	(*paste*)

Damit lassen sich beliebige Mengen an Text bearbeiten:

Text	kopieren	ändern	löschen
wortweise vorwärts	`yw`	`cw`	`dw`
wortweise rückwärts	`yb`	`cb`	`db`
bis zum Zeilenanfang	`y0`	`c0`	`d0`
bis zum Zeilenende	`y$`	`c$` oder `C`	`d$` oder `D`
die ganze Zeile	`yy`	`cc`	`dd`

Tip: Mir erscheint es logisch, mit `Y` alles bis zum Zeilenende zu kopieren. Da dies nicht standardmäßig der Fall ist, habe ich mir ein eigenes Mapping in der *Konfigurationsdatei* so definiert.

Natürlich lassen sich die Anweisungen wieder beliebig multiplizieren, `c3W` oder `3cW` ändert die nächsten drei Wörter ohne Rücksicht auf Satzzeichen, `y3y` oder `3yy` löscht die nächsten drei Zeilen. Bei umfassenderen Textmengen empfiehlt es sich, diese zuerst im *visuellen Modus* zu markieren, und dann die entsprechende Taste für die gewünschte Bearbeitungsfunktion zu drücken.

Will man nur einzelne Buchstaben oder Ziffern abändern, so kann man folgende Funktionen nutzen:

`x`	Lösche das Zeichen unter dem Cursor	
`~`	Ändere Kleinbuchstaben in Großbuchstaben und umgekehrt	
`gu`	Ändere Buchstaben in Kleinbuchstaben (auch visuell markierte Bereiche)	
`gU`	Ändere Buchstaben in Großbuchstaben (auch visuell markierte Bereiche)	
`r`	Ändere das Zeichen unter dem Cursor, danach weiter im Normal-Mode	(*replace*)
`R`	Überschreibe eine beliebige Anzahl an Zeichen („Replace“-Mode, zurück mit `ESC`)	
`s`	Ändere das Zeichen unter dem Cursor, weiter im Insert-Mode	(*substitute*)
`S`	Ändere die ganze Zeile	

Bei jeder Bearbeitungsanweisung wird der entsprechende Textteil in die Zwischenablage kopiert. Von dort aus kann er mittels `p` wieder eingefügt werden:

`p`	Füge Inhalt des Zwischenspeichers *hinter* dem Cursor ein
`P`	Füge Inhalt des Zwischenspeichers *vor* dem Cursor ein

Im Einfügemodus kann Text aus der systemweiten Zwischenablage mittels `Shift Insert` (Einfüge-Taste) oder durch Klick auf die mittlere Maustaste (gleichzeitiges Klicken von linker und rechter Taste bei zweitastigen Mäusen und Notebooks) eingefügt werden.

Im Normalmodus kann Text aus der systemweiten Zwischenablage mittels des *Registers* `*` genutzt, also mittels `"*p` beziehungsweise `"*P` eingefügt werden.

Undo und Redo

Änderungen können mit `u` rückgängig gemacht beziehungsweise mit `Ctrl r` wiederhergestellt werden:

`u`	Mache die letzte Änderung rückgängig	(*undo*)
`U`	Mache alle Änderungen in der aktuellen Zeile rückgängig	
`Ctrl r`	Stelle eine rückgängig gemachte Änderung wieder her	(*redo*)
`.`	Wiederhole die zuletzt getätigte Texteingabe, Textbearbeitung, Formatierung, etc.	

Marker

Muss man öfters innerhalb einer Datei hin- und herspringen, so schaffen Markierungshilfen (*Marker*) Abhilfe.

Im Normalmodus kann man die Stelle, an der sich der Cursor gerade befindet, mit `m` gefolgt von einem beliebigen Buchstaben markieren:

`m Kleinbuchstabe`	Setze eine lokale Markierung (gilt nur in der aktuellen Datei)
`m Großbuchstabe`	Setze eine globale Markierung

Mit Hilfe der globalen Markierungen lassen sich häufig genutzte Dokumente schnell laden, egal wo man sich gerade befindet.

Beispiel: Man kann man mit `'G` zu genau der Stelle wechseln, die man vorhergehend mit `mG` markiert hat. Liegt der Marker dabei in einer anderen Datei, so bleibt der ursprüngliche Buffer im Hintergrund geöffnet (ein *Wechsel zwischen den geöffneten Dateien* ist beispielsweise mit dem *Minibuf-Explorer*-Plugin leicht möglich).

Mit `'` (einfaches Anführungszeichen), gefolgt von dem angegebenen Buchstaben, gelangt man von wieder zu der entsprechenden Zeile, mittels `` ` `` (Apostroph) sogar in die entsprechende Spalte. Mittels `''` beziehungsweise `` `` `` gelangt man zur zuletzt bearbeiteten Zeile beziehungsweise Position zurück.

Tipp: Mittels `'.` gelangt man zu der zuletzt editierten Stelle, mit `'^` zur letzten Einfüge-Stelle, und mit `'"` zur Position beim letzten Beenden zurück!

Ein weiteres Springen zwischen verschiedenen Änderungen und deren Positionen ist mittels `Ctrl o` beziehungsweise `Ctrl i` möglich:

`Ctrl o`	Gehe zurück zur letzten Änderung (beziehungsweise zur zuletzt geänderten Datei)
`Ctrl i`	Gehe vorwärts zur letzten Änderung (umgekehrte Richtung)

`Ctrl o` beziehungsweise `Ctrl i` können auch mehrfach hintereinander gedrückt werden, um weiter zurück beziehungsweise vor in der Positions-History zu springen.

Register

Vim besitzt nicht nur *eine* Zwischenablage, sondern kann Textelemente und *Makros* jedem beliebigen Kleinbuchstaben zuweisen. Ein Register ist quasi eine benannte Zwischenablage.

Im Normalmodus kann man mit `" Buchstabe` auf einen Register zugreifen:

`" Kleinbuchstabe Bearbeitungsanweisung`	Kopiere in/aus das Register `Buchstabe` hinein/heraus
`" Großbuchstabe Bearbeitungsanweisung`	Füge Text oder Code hinten an das Register `Buchstabe` an

Beispiel: Mittels `"hyy` kann die aktuelle Zeile in die Ablage `h` kopiert werden. Deren Inhalt kann mit `"hp` wieder an anderer Stelle eingefügt werden. So abgelegte Inhalte bleiben auch beim Schließen von Vim erhalten!

Mit `:reg` erhält man eine Übersicht, welcher Inhalt in welchem Register abgelegt ist:

`:reg`	Zeige den Inhalt aller Register an

Die normale, Vim-interne (namenlose) Zwischenablage kann explizit über das Register `"` angewählt werden. Ein anderes, spezielles Register ist die systemweite Zwischenablage `*`, mittels der ein Kopieren von beziehungsweise in andere(n) Programme(n) möglich ist:

`"*y Bewegung`	Kopiere in die Zwischenablage
`"*p Bewegung`	Füge aus der Zwischenabage ein

Eine weitere systemweite Zwischenablage ist `~`, in welcher der zuletzt mit der Maus ausgewählte Text abgespeichert ist.

Unter Linux werden Bereiche bereits durch ein einfaches Markieren (*Visueller Modus*) in die systemweiten Zwischenablagen kopiert. An anderer Stelle können sie dann mit `Shift Ins` (Einfüge-Taste) oder durch einen Klick auf die mittlere Maustaste wieder eingefügt werden.

Auch im Einfüge- und im Kommandozeilenmodus kann auf die Inhalte der einzelnen Register zugegriffen werden: Durch eine aufeinander folgende Eingabe von `<c-r>` und einem Registerbuchstaben beziehungsweise -Symbol wird der Inhalt des entsprechenden Registers an der Cursorposition eingefügt:

`<c-r>"`	Füge den Inhalt des namenlosen Registers an der Cursorpostition ein
`<c-r>%`	Füge den aktuellen Dateinamen (Register `%`) an der Cursorposition ein
`<c-r>a`, `<c-r>b` usw.	Füge den Inhalt des Registers `a`, `b`, usw. an der Cursorposition ein

Weitere Informationen über die verschiedenen Register aus den Vim-Hilfeseiten können mittels `:h <c-r>` aufgerufen werden.

Makros

Es kann nicht nur Text in einem *Register* abgelegt werden, sondern auch jede beliebige Anweisungssequenz. Wie bei einem Kassettenrecorder können Anweisungen mit aufgezeichnet, und als „Makro“ später beliebig oft wieder abgespielt werden:

Im Normalmodus werden Makros mit `q Buchstabe` aufgezeichnet und mit `@ Buchstabe` wiedergegeben:

`q Kleinbuchstabe`	Nehme eine Anweisungssequenz bis zum nächsten Drücken von `q` auf
`q Großbuchstabe`	Hänge eine Anweisungssequenz an das Register `Buchstabe` an
`@ Buchstabe`	Führe die im Register `Buchstabe` liegende Anweisungssequenz aus

Es kann durchaus nützlich sein, z.B. mittels `10@Buchstabe` eine Anweisungskette 10fach auszuführen. Speziell gleichförmige Bearbeitungen mehrerer Dateien sind so möglich, denn *Bufferwechsel* können ja gleich mit „aufgenommen“ werden.. :-)

Tipp: Der zuletzt ausgeführte Makro-Anweisung kann mit `@@` wiederholt werden.

Faltungen

Werden Text-Dateien infolge ihrer Länge zu unübersichtlich, können bestimmte Bereiche ausgeblendet werden. Das kann entweder über Schlüsselworte, Einrückungen oder über Symbole erfolgen.

Beispielsweise wird Python-Code häufig anhand von Einrückungen gefaltet, in Datei-Vergleichen mittels *diff* werden gleiche Textbereiche weggefaltet, so dass nur die Unterschiede in den Dateien sichtbar bleiben. Wird kein spezieller Faltungsmechanismus von einem Plugin geladen, so wird das in der *Konfigurationsdatei* festgelegt Faltungsschema verwendet. Oft werden dabei als Faltungsmarkierungen `{{{` und `}}}` genutzt, so dass alle Textbereiche, die sich zwischen solchen Dreifach-Klammern befinden, gefaltet werden.

Folgende Anweisungen können im Umgang mit Faltungen nützlich sein:

`zf`	Erstelle eine Faltung	
`zo`	Öffne eine Faltung	(*open*)
`zc`	Schließe eine Faltung	(*close*)
`zd`	Entferne eine Faltung	(*delete*)
`\rf`	Falte die Datei neu	(*refold*)

Um eine Faltung zu erstellen, wird der Bereich meist zuerst visuell markiert, und dann mittels `zf` gefaltet.

Faltungen können auch ineinandergeschachtelt (*nested*) auftreten. Faltungen unter dem Cursor können einzeln oder auf einmal mittels `za` beziehungsweise `zA` geöffnet und geschlossen werden.

`za`	Öffne beziehungsweise schließe lokale Faltungen
`zA`	Öffne beziehungsweise schließe lokale Faltungen (rekursiv)

Ebenfalls nützlich sind folgende Faltungsanweisungen:

`zr`	Reduziere die Anzahl der Faltungsebenen um eins	(*reduce*)
`zm`	Erhöhe die Anzahl der Faltungsebenen um eins	(*more*)
`zR`	Öffne alle Faltungen	
`zM`	Schließe alle Faltungen	

Persönlich habe ich mir in der Konfigurationsdatei mittels `nmap <space> za` ein Mapping definiert, das beim Drücken der Leertaste `za` aufruft, und damit die aktuelle Textebene faltet oder aber , falls man sich mit dem Cursor über einer Faltung befindet, diese wiederum öffnet.

Fenster splitten

Vim kann mehrere Dateien optional in verschiedenen Tabs im oder in unterteilten Fenstern öffnen:

- Mit `:tabedit datei` wird eine Datei in einem neuen Tab geöffnet. Zwischen den Tabs kann mit `Ctrl PageUP` und `Ctrl PageDOWN` gewechselt werden. Infos findet man beispielsweise unter `:h tabpage.txt`.[2]
- Mit `:[v]split` beziehungsweise `Ctrl W s` oder `Ctrl W v` wird ein Fenster horizontal beziehungsweise vertikal geteilt. Manche Plugins wie *Minibuf-Explorer* oder *Tagbar* nutzen diese Funktion, um auf der linken Seite beispielsweise ein Inhaltsverzeichnis ein- oder auszublenden.

Anweisungen zur Handhabung von geteilten Fenstern werden gewöhnlich mit `Ctrl W` eingeleitet. Mit folgenden Tastenkombinationen kann man zwischen den geöffneten Fenstern wechseln:

[2] Persönlich nutze ich anstelle von Tabs lieber verschiedene *Buffer*, insbesondere in Kombination mit dem *Minibuf-Explorer*.

`Ctrl W w`	Wechsle zum jeweils nächsten Fenster (im Uhrzeigersinn)
`Ctrl W h j k l`	Wechsle man zum nächsten Fenster auf der linken, unteren, oberen oder rechten Seite
`Ctrl W H J K L`	Verschiebe das aktuelle Fenster in die jeweilige Richtung
`Ctrl W r` bzw. `Ctrl W R`	Verschiebe alle geöffnete Fenster der Reihenfolge nach, das letzte wird das erste

Mit folgenden Anweisungen lässt sich die Größe des aktuellen Fensters anpassen:

`Ctrl W +`	Vergrößere das aktuelle Fenster um eine Zeile
`Ctrl W -`	Verkleinere das aktuelle Fenster um eine Zeile
`n Ctrl W |`	Setze die Breite des aktuellen Fensters auf `n`
`Ctrl W _`	Maximiere das aktuelle Fenster
`Ctrl W =`	Richte alle Fenster auf die gleiche Größe aus

Zum Schließen des aktuellen beziehungsweise der übrigen Fenster gibt es folgende Tastenkombinationen:

`Ctrl W c`	Schließe das aktuelles Fenster	(*close*)
`Ctrl W o`	Schließe alle anderen Fenster	(*close other*)

Quickfixleiste

Nutzt man den Vim als Programmier-Umgebung beziehungsweise compiliert aus dem Vim heraus Quellcode, so bekommt man Fehlermeldungen in der sogenannten „Quickfix-Leiste“ angezeigt. Im Prinzip ist das ein gesplittetes Fenster, in welchem zwischen den Fehlern navigiert werden kann. Durch Drücken von `Enter` gelangt man an die entsprechende Stelle im Hauptdokument. Von dort aus gelangt man zum nächsten beziehungsweise vorherigen Fehler mittels `:cn` beziehungsweise `:cp`.

Gesplittete Fenster werden häufig von Plugins genutzt, um beispielsweise am linken Seitenrand ein *Inhaltsverzeichnis* oder am unteren einen *Buffer-Browser* einzublenden.

Vimdiff

Das Linux-Programm `vimdiff` zeigt ebenfalls in gesplitteten Fenstern Unterschiede zwischen zwei Dateien an. Auf diese Weise lassen sich verschiedenen Versionen des gleichen Dokuments schnell und übersichtlich abgleichen (abweichende Stellen werden automatisch markiert):

```
vimdiff datei1 datei2
```

Bewegt man sich in einer Datei nach unten, so scrolled die Anzeige der anderen Datei im gegenüberliegenden Fenster mit, so dass stets die entsprechenden zwei Zeilen verglichen werden. Beide Dateien können editiert werden, der Abgleich erfolgt automatisch.

Kommandozeilen-Modus

Im Kommandozeilen-Modus können sowohl Vim-Anweisungen als auch *externe Systemanweisungen* aufgerufen werden.

Ausgehend vom Normalmodus gelangt man mittels `:` in den Kommandozeilen-Modus, mittels `Esc` wieder zurück.

Buffer öffnen

Eine neue Datei kann mit `:edit` (oder kurz `:e`) als neuer Buffer geöffnet werden, wobei die bisherige Datei als eigener Buffer im Hintergrund geladen bleibt.

`:e dateiname`	eine Datei öffnen beziehungsweise neu erstellen
`:e %`	die aktuelle Datei (`%`) neu laden

Das Zeichen `%` hat im Kommandozeilen-Modus eine eigene Bedeutung: Es steht für den Namen der aktuell geöffneten Datei. Die Ausgabe von `%` lässt sich unter anderem folgendermaßen ändern:

`%`	Aktueller Dateiname
`%:~`	Aktueller Dateiname (relativ zum Home-Verzeichnis)
`%:p`	Aktueller Dateiname (absoluter Pfad)
`%:p:h`	Aktuelles Verzeichnis („Head“ des absoluten Pfads)
`%:p:h:h`	Übergeordnetes Verzeichnis („Head“ des aktuellen Verzeichnisses)
`%:p:r`	Aktueller Dateiname ohne Endung (absoluter Pfad)
`%:e`	Endung der aktuellen Datei

Eine vollständige Beschreibung kann mittels `:h filename-modifiers` aufgerufen werden.

Der Inhalt einer anderen Datei kann mittels `:read` (oder kurz `:r`) hinter der momentanen Cursor-Position eingefügt werden:

`:r dateiname`	Inhalt einer Datei vor dem Cursor einfügen

Buffer speichern und/oder wechseln

Um Vim zu beenden oder einzelne Dateien beziehungsweise Fenster zu schließen, gibt man folgendes ein:

`:q`	Schließe den aktuellen Buffer schließen (und beende Vim, falls nur ein Fenster offen ist)
`:qa`	Schließe alle Buffer, beende Vim
`:q!`	Schließe den aktuellen Buffer, verwerfe ungespeicherte Änderungen
`:w`	Speichere Änderungen an der aktuellen Datei
`:wq`	Speichere die aktuelle Datei und schließe den Buffer
`:wqa`	Speichere alle geöffneten Dateien und beende Vim
`:w Dateiname`	Speichere den aktuellen Buffer als `Dateiname`

Auch die Schreib-Anweisung kann nur auf einen bestimmten Textbereich angewendet werden. Beispielsweise kann man mit `:1,25w dateiname` die ersten `25` Zeilen der Datei in den angegebenen Dateinamen schreiben, oder man kann einen Text visuell markieren und dabei `:w` eingeben; in der Kommandozeile wird dabei automatisch `'<,'>w` angezeigt, wobei `'<,'>` den visuell markierten Bereich bezeichnet.

In Vim können mehrere Dateien auf einmal geöffnet sein. Im Umgang mit diesen Buffern sind folgende Anweisungen hilfreich:

`:ls`	Zeige eine Liste an geöffneten Dateien an	(identisch mit `:buffers`)
`:bn`	Gehe zur nächsten offenen Datei	(*buffer next*)
`:bp`	Gehe zur vorherigen offenen Datei	(*buffer previous*)
`:bf`	Gehe zur ersten geöffneten Datei	(*buffer first*)
`:bl`	Gehe zur letzten geöffneten Datei	(*buffer last*)
`:b#`	Gehe zur zuletzt verwendeten Buffer	
`:b 1..99`	Gehe zur Datei Nr. `1..99` der Bufferliste	
`:bd`	Lösche die aktuelle Datei aus der Buffer-Liste	(*buffer delete*)

Mittels `:bufdo` kann man eine (oder mehrere mittels `|` („Pipe“) verknüpfte) Anweisung(en) auf alle geöffneten Dateien anwenden:

`:bufdo Anweisung`	Führe eine `Anweisung` in allen geöffneten Buffern aus

Das Gleiche kann man allerdings auch (meist sicherer) mittels einer *Makro-Aufzeichnung* erreichen.

Externe Programme aufrufen

Externe Programme können im Kommandozeilen-Modus integriert werden, indem dem jeweiligen Aufruf ein `!` vorangesetzt wird, wie zum Beispiel:

`:ls`	Zeige eine Liste der geoffneten Buffer an (Vim-interne Funktion!)
`:!ls`	Gebe den Inhalt des Arbeitsverzeichnisses aus (gewöhnliche Linux-Anweisung)
`:.!sh`	Ersetze die momentane Zeile (`.`) durch die Rückgabe der Shell

Wird ein beliebiger Bereich vor dem Ausrufezeichen angegeben (z.B. `%` für die aktuelle Datei oder `.` für die aktuelle Zeile), so wird die Rückgabe der aufgerufenen Anweisung an entsprechender Stelle in die Datei geschrieben.

Die Ausgabe eines externen Programms kann wiederum mittels `:r` unmittelbar hinter der aktuellen Cursorposition eingelesen werden. Um beispielsweise die Ausgabe von `!ls` in den Buffer aufzunehmen, kann man folgendes eingeben:

```
:r !ls
```

Im Normalmodus kann auch `!!` anstelle von `:!` eingegeben werden, um externe Programme aufzurufen.

Externe Programme können unter anderem eingesetzt werden, um die aktuelle Datei zu compilieren. Um beispielsweise eine aktuell geöffnete LaTeX -Datei in ein `.pdf`-Dokument umzuwandeln, kann man folgendes eingeben:

```
:!pdflatex %
```

Für längere derartige Aufrufe können natürlich wiederum in der *Konfigurationsdatei* entsprechende *Mappings* vergeben werden. Danach genügt im Normal- oder Einfügemodus ein individuelles Tastenkürzel, und der dadurch definierte Prozess wird ausgeführt.

Ebenso ist es möglich, externe Programme nur auf einen bestimmten Bereich (z.B. im *visuellen Modus*) anzuwenden:

`'<,'> !sort`	Sortiere den visuell markierten Bereich (`'<` bis `'>`)
`'a,'b !grep Wort`	Lösche alle Zeilen zwischen den Markern `a` und `b`, die nicht `Wort` enthalten
`:r !grep "Test" Datei`	Lese die Ausgabe von `grep` ein und füge sie nach der aktuellen Stelle in die `Datei` ein

Das externe Programm muss also nicht an erster Stelle in der Kommandozeile erscheinen.

Text ersetzen

Gezieltes Ersetzen von Text erfolgt in Vim nach folgendem Schema:

```
:Bereich s/Suchbegriff/Ersetzung/Optionen
```

Als Optionen stehen dabei zur Verfügung:

`c`	Frage bei jedem Treffer nach	(*confirmation*)
`g`	Beachte alle Vorkommen des Suchbegriffs (nicht nur den ersten Treffer in jeder Zeile)	(*global*)
`i`	Ignoriere Groß- / Kleinschreibung	(*ignore case*)

Wird eine dieser Anweisungen auf einen visuell markierten Bereich angewandt, so werden dessen Grenzen `'<`, `'>` als Bereich angenommen. Ansonsten kann jeder beliebige Zeilenbereich, mit Komma getrennt, angegeben werden. Möchte man Ersetzungen in der ganzen Datei vornehmen, so steht dafür `%` als Auswahlbereich zur Verfügung.

Beispiel:

`:% s/alt/neu/g`	Ersetze `alt` durch `neu` in der ganzen Datei
`:1,20 s/alt/neu/g`	Ersetze `alt` durch `neu` in den ersten 20 Zeilen

Kommt der Schrägstrich selbst im Suchbegriff vor, kann auch jedes andere Zeichen zur Trennung von Suchbegriff, Ersetzungen und Optionen gewählt werden. Das erste Zeichen nach dem `s` wird dann als Trennzeichen verwendet (z.B. `:%s #/pfad/#irgendwas#`).

Bisweilen ist es auch hilfreich, „seltsame“ Zeichen in einer Textdatei zu ersetzen, beispielsweise wenn Text aus einer `.pdf`-Datei mittels `pdftotext` in eine Textdatei extrahiert wird. Die zu löschenden Zeichen können dann visuell markiert und mittels `y` in die Vim-interne Zwischenablage kopiert werden. In der Kommandozeile kann der so kopierte Inhalt dann mittels `<c-r> *` wieder eingefügt werden.

Reguläre Ausdrücke

Das Suchen und Ersetzen von Textstücken lässt sich durch so genannte reguläre Ausdrücke oft wesentlich erleichtern beziehungsweise beschleunigen. Hierzu können spezielle Zeichen verwendet werden, die jeweils einem bestimmten Suchmuster entsprechen.

Werden die folgenden Zeichen in einer Such- oder Ersetzungsanweisung verwendet, so werden sie als reguläre Ausdrücke interpretiert. Möchte man das jeweilige Zeichen in seiner Grundbedeutung interpretiert haben, so muss ein \ (Backslash) davor platziert werden:

`\`	Sonderbedeutung des nächsten Zeichens aufheben („" entspricht einem Backslash)
`^`	Zeilenanfang
`$`	Zeilenende
`\r`	Zeilenende (carriage return)
`\t`	Tabulator
`.`	Ein beliebiges Zeichen
`*`	Multiplexer: Das vorhergehende Zeichen null mal oder beliebig oft
`[ ]`	Selektierer: Eines der Zeichen innerhalb der eckigen Klammern
`[^ ]`	Selektierer mit Negation: Ein Zeichen, das *nicht* in der eckigen Klammer vorkommt
`&`	Nur im Ersetzungsbereich: Textstelle, auf die das Suchmuster zutrifft.

Ebenso gibt es Zeichen, die in einer Such- oder Ersetzungsanweisung als „normale“ Zeichen interpretiert werden, jedoch durch Voranstellen eines \ eine Sonderbedeutung bekommen:

`\<`	Wortanfang
`\>`	Wortende
`\(` `\)`	UND-Verknüpfung: Gruppierung mehrer Suchmuster zu einem Ausdruck
`|`	ODER-Verknüpfung: Der links oder der rechts von `|` stehende Ausdruck
`\_.`	Ein beliebigs Zeichen, auch Zeilenende-Zeichen (Suche über Zeilenumbrüche hinweg)
`\+`	Multiplexer: Das vorhergehende Zeichen einmal oder beliebig oft.
`\?`	Multiplexer: Das vorhergehende Zeichen null oder ein mal.

Teile eines regulären Ausdrucks, die beim Suchen mittels `\(` und `\)` gruppiert werden, können im neuen Ausdruck mittels `\1`, `\2`, `\3` usw. wieder aufgegriffen werden, wobei beispielsweise `\1` den ersten gruppierten Ausdruck bezeichnet. Die Textstelle, die beim Suchen auf den *gesamten* regulären Ausdruck zutrifft, kann beim Ersetzen mittels `\0` referenziert werden.

Visueller Modus

Im visuellen Modus kann Text mittels *Bewegungsanweisungen* markiert werden, um ihn zu kopieren oder zu bearbeiten.

Aus dem Normal-Modus gelangt man wie folgt in den visuellen Modus:

`v`	„normaler“ visueller Modus	
`V`	zeilenweise visueller Modus	(*Visual Line*)
`Ctrl v`	spaltenweise visueller Modus	(*Visual Block*)

Im normalen visuellen Modus wird der gesamte Textbereich von der aktuellen Position aus bis zu der Stelle, zu der man sich hinbewegt, markiert. Mit `o` („other“) kann man an das andere Ende des visuell markierten Bereichs gelangen.

Im zeilenweise-visuellen Modus können mit den Navigationsanweisungen `{` und `}` oder mit Hilfe von Markierungen leicht ganze Paragraphen oder Textabschnitte kopiert, verschoben oder anderweitig bearbeitet werden. Der blockweise-visuelle Modus bietet speziell mit dem Vim-Plugin Align eine elegante Möglichkeit zur Bearbeitung von Spalten einer Tabelle.

Tipp: Um Zeilen mit verschieden langen Inhalten am Ende mit Leerzeichen aufzufüllen, um beispielsweise dahinter weiteren Text als eine neue Spalte einfügen zu können, können jeweils die ersten Zeichen der Zeilen im visuellen Blockmodus markiert werden und anschließend `$` gedrückt werden. Es wird dadurch der gesamte Text markiert. Gibt man dann `A <Leertaste>` ein, so wird am Ende der längsten Zeile ein Leerzeichen angefügt und alle kürzeren Zeilen bis zur gleichen Breite mit Leerzeichen aufgefüllt.

Weiterhin gibt es im visuellen Modus Mappings für folgende Auswahl-Kriterien, die wahlweise mit `i` („inner“) ohne umgebende Whitespaces oder mit `a` („outer“) mitsamt umgebenden Whitespaces eingeleitet werden können:

`w`	Ein einzelnes Wort (siehe Option `iskeyword`)
`s`	Ein einzelner Satz
`p`	Ein einzelner Paragraph (Absatz)
`t`	Ein HTML/XML-Tag
`"`, `'`, `` ` ``	Durch Anführungszeichen begrenzter Text
`{`, `[`, `(`, `<`	Durch Klammern begrenzter Text

Gibt man also beispielsweise im visuellen Modus `ip` ein, so wird der aktuelle Absatz (ohne vorangehende und darauf folgende Leerzeile) ausgewählt; ebenso kann Text innerhalb zwei runder Klammern mit `a(` inklusive der Klammern ausgewählt werden, wenn sich der Cursor innerhalb der Klammern befindet.

Tip: Jede Bearbeitungsanweisung, die für gewöhnlich eine darauffolgende Bewegungs- oder Auswahlanweisung erwartet, kann auch direkt einen markierten Bereich angewandt werden.

Konfigurationsdatei

In der Konfigurationsdatei *~/.vimrc* können zahlreiche Optionen wie z.B. Zeilennummerierungen, Markierungen von Suchergebnissen, automatische Backups etc. an- oder ausgeschaltet werden. Gleichzeitig können individuelle Tastenkombinationen für beliebige Funktionen oder Funktionsketten festgelegt werden.

Meine persönliche Konfigurationsdatei sieht etwa so aus:

`Beispieldatei ~/.vimrc`

Mappings

Mappings ermöglichen es, für beliebige Anweisungen selbst gewählte Tastenkombinationen zu vergeben.

Je nach Modus, in welchem die Mappings gelten sollen, wird zwischen `map`, `imap`, und `vmap` unterschieden:

`nmap Kürzel Befehl`	Kürzel für den Normal-Modus
`cmap Kürzel Befehl`	Kürzel für den Kommando-Modus
`imap Kürzel Befehl`	Kürzel für den Einfüge-Modus
`vmap Kürzel Befehl`	Kürzel für den Visuell-Modus

Mappings können auch über den Kommandozeilen-Modus gesetzt werden, bleiben so allerdings nur bis zum Beenden von Vim gespeichert. Für häufig genutzte Tastenkombinationen empfiehlt es sich daher, diese in der Konfigurationsdatei festzuhalten.

Tipp: Eine Übersicht darüber, welche Mappings vergeben sind, kann man mit `:map` bekommen. Um die Auswahl einzuschränken, kann dahinter der Beginn eines Mappings stehen - z.B. listet `:map \b` alle Mappings auf, die mit `\b` beginnen. In entsprechender Weise zeit `:imap \b` alle Mappings im Einfügemodus an, die mit `\b` beginnen.

Folgende Sonderzeichen werden in Mappings häufig verwendet, wobei die Groß- und Kleinschreibung keine Rolle spielt:

`<Leader>`	\ (Backslash)
`<Return>` oder `<CR>`	`Enter`-Taste (Carriage Return)
`<Esc>`	`Escape`-Taste
`<Bar>`	| (Pipe-Symbol)
`<Space>`	Leerzeichen beziehungsweise Leertaste
`<Tab>`	Tabulator
`<C-A>`	`Ctrl A`

Rechtschreibprüfung

Seit der Version 7.0 bietet Vim von sich aus eine integrierte Rechtschreibprüfung. Sie wird allgemein durch ein Eingabe folgender Art in der Vim-Kommandozeile aktiviert:

```
:set spell spelllang=en_us
```

Bei aktiver Rechtschreibprüfung werden Wörter als „falsch“ angesehen, wenn es dazu keinen Eintrag in der entsprechenden Spell-File gibt. Als Name der Sprache wird allgemein die Form `language_region` verwendet, da sich beispielsweise die in Großbritannien übliche Rechtschreibung `en_en` von der amerikanischen Rechtschreibung `en_us` unterscheidet. Für die deutschsprachige Rechtschreibung gibt es neben `de_de`, `de_at` (Österreich) und `de_ch` (Schweiz) auch `de_19` und `de_20` für die alte beziehungsweise neue deutsche Rechtschreibung.

Wird eine Sprachdatei bei Aktivierung mittels `:set spell` nicht gefunden, so wird automatisch ein vorinstalliertes Plugin namens `spellfile.vim` gestartet, das es dem Benutzer anbietet, die entsprechende Sprachdatei automatisch herunterzuladen und zu aktivieren. Falsch geschriebene Wörter werden dann (je nach Farbschema) durch eine rote Hintergrundfarbe gekennzeichnet.

Mit `:set nospell` kann die Rechtschreibprüfung wieder abgeschaltet werden. Noch einfacher wird das Aktivieren und Deaktivieren der Rechtschreibung durch die Aufnahme der folgenden Funktion (nach einer Vorlage aus dem Vim-Wiki) in der Konfigurationsdatei `~/.vimrc`:

```
" Rechtschreibprüfung mit <F8> an- und ausschalten:
let g:myLang = 1
let g:myLangList = ['nospell', 'de_20,en_us']
function! MySpellLang()
  if g:myLang == 0 | setlocal nospell | endif
  if g:myLang == 1 | let &l:spelllang = g:myLangList[g:myLang] | setlocal spell␣
↪| endif
  echomsg ''
  let g:myLang = g:myLang + 1
  if g:myLang >= len(g:myLangList) | let g:myLang = 0 | endif
endfunction
```

```
nmap <F8> :call MySpellLang()<CR>
imap <F8> <C-o>:call MySpellLang()<CR>"
```

Mit dieser Funktion kann durch Drücken von `<F8>` die Rechtschreibprüfung an- und ausgeschaltet werden. Durch die Auswahl `'de_20,en_us'` an Sprachdateien sind sowohl die neue deutsche als auch die US-amerikanische Rechtschreibung aktiv; ein Wort wird somit nur dann als „falsch“ durch ein Highlighting hervorgehoben, wenn es in keiner der beiden Sprachdateien vorkommt.

Bei aktiver Rechtschreibprüfung können unter anderem die folgenden Tastenkombinationen verwendet werden:

`]s`	Gehe zum nächsten falschen Wort
`[s`	Gehe zum vorherigen falschen Wort
`zg`	Füge das Wort unter dem Cursor dem aktuellen Wörterbuch hinzu (Variable: `spellfile`)
`zG`	Speichere das Wort unter dem Cursor in einer internen Wortliste; diese wird nach dem Schließen von Vim gelöscht
`zw`	Speichere das Wort unter dem Cursor als „falsch“ in der aktuellen Wörterbuchdatei (Variable: `spellfile`)
`zW`	Speichere das Wort unter dem Cursor als „falsch“ in der internen Wortliste
`zug`, `zuw`, `zuG`, `zuW`	Lösche das Wort unter dem Cursor aus der entsprechenden Liste

Eine ausführliche Hilfe erhält man mittels `:h spell.txt`.

Plugins

Vim kann durch so genannte „Plugins“ erweitert werden. Diese können von der Projektseite oder von Github heruntergeladen werden.

Unterschieden wird zwischen allgemeinen Plugins, die nach ihrer Installation automatisch geladen werden, und „Filetype“-Plugins, die nur geladen werden, wenn eine Datei des entsprechenden Typs geladen wird. Gewöhnlich erkennt Vim den Dateityp anhand der Endung, z.B. `datei.py` als eine `Python`-Quellcode-Datei. Der Dateityp einer geöffneten Datei kann allerdings auch manuell mittels `set filetype Typ` geändert werden, wobei die entsprechenden Filetype-Plugins nachgeladen werden.

Vundle: Plugins installieren und verwalten

Das Vundle-Plugin bietet eine sehr empfehlenswerte Methode, eine individuelle Auswahl an Plugins zu installieren und zu verwalten. Um dieses Plugin zu verwenden, wird zunächst im Ordner `~/.vim` ein neues Unterverzeichnis namens `bundle` angelegt:

```
mkdir ~/.vim/bundle/
```

Anschließend gibt man folgende Zeile ein:

```
git clone https://github.com/gmarik/Vundle.vim.git ~/.vim/bundle/Vundle.vim
```

Damit ist das Vundle-Plugin installiert. Die Benutzung des Plugins funktioniert mittels der Konfigurationsdatei `~/.vimrc`. Hier wird an beliebiger Stelle folgendes eingegeben:

```
set nocompatible

" Set the runtime path and initialize Vundle:
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()

" Let Vundle manage itself (required)
Plugin 'gmarik/Vundle.vim'

" Install and use the following Plugins:
Plugin 'vim-scripts/sudo.vim'
Plugin 'scrooloose/nerdcommenter'
Plugin 'SirVer/ultisnips'

" ...

filetype on                         " Filetype-Erkennung aktivieren
filetype indent on                  " Syntax-Einrückungen je nach Filetype
filetype plugin on                  " Filetype-Plugins erlauben
```

Hierbei muss lediglich beachtet werden, dass die `filetype on`-Anweisungen erst **nach** den für Vundle relevanten Zeilen eingegeben werden.

Um ein Plugin mittels Vundle zu installieren, genügt es den GitHub-Namen des Plugins mittels `Plugin 'repository/plugin-name'` in die Vundle-Liste aufzunehmen. Anschließend wird Vim gestartet und `:PluginInstall` eingegeben. Alle noch nicht installierten Plugins werden damit automatisch als eigene Verzeichnisse in das `~/.vim/bundle`-Verzeichnis installiert.

Mit Vundle ist auch eine weitere Verwaltung der Plugins möglich:

- Um ein Plugin zu deaktivieren, genügt es die entsprechende `Plugin`-Zeile in der `~/.vimrc` auszukommentieren und Vim neu zu starten; es werden nämlich bei Verwendung von Vundle nur diejenigen Plugins von Vim geladen, die in der Plugin-Liste enthalten sind.
- Soll ein Plugin entfernt werden, so wird es ebenfalls zunächst auskommentiert, Vim anschließend neu gestartet und `:PluginClean` eingegeben.
- Um alle Plugins auf einmal zu aktualisieren, kann man `:PluginInstall!` eingeben. Vundle prüft damit automatisch, ob auf GitHub eine neuere Version des Plugins existiert und installiert diese gegebenenfalls nach.

  Soll ein Plugin bei einer Aktualisierung mittels `:PluginInstall!` **nicht** berücksichtigt werden, so kann bei diesem als zusätzliche Option `{"pinned, 1}` angegeben werden, beispielsweise:

```
Plugin 'honza/vim-snippets', {'pinned': 1}
```

Wird ein Plugin auf diese Weise eingebunden, so werden beispielsweise eigene Änderungen durch einen Update nicht überschrieben.

Nahezu jedes Vim-Plugin wird inzwischen entweder vom jeweiligen Entwickler oder innerhalb des `vim-scripts`-Repository auf GitHub gelistet. Bei den folgenden Beschreibungen der einzelnen Plugins sind daher neben den Beschreibungen auf der Vim-Projektseite stets auch die entsprechenden GitHub-Repositories verlinkt.

Hilfreiche Erweiterungen

Align

Das Align-Plugin stellt eine gleichnamige Funktion bereit, mittels derer man visuell markierte Bereiche zu einer übersichtlichen Tabelle ausrichten kann.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/jezcope/vim-align

Als Anwendungsbeispiel sei in einer Textdatei folgende Tabelle enthalten:

```
# Vorher:
a ; b ; c ; d ; e;
ab; bc; cd; de; ef;
abcd ; bcde ; cdef ; defg ; efgh;
```

Nach einer visuellen Markierung des Textes und Eingabe von `:Align ;` sieht die Datei so aus:

```
# Nachher:
a    ; b    ; c    ; d    ; e    ;
ab   ; bc   ; cd   ; de   ; ef   ;
abcd ; bcde ; cdef ; defg ; efgh ;
```

Die `:Align`-Funktion akzeptiert jedes beliebige Trennzeichen und kann entweder global oder mittels `:'<,'>Align` auf den aktuell markierten Bereich angewendet werden. Die Bearbeitung von tabellarischem Text wird so wesentlich erleichtert. :-) Das Calendar-Plugin ermöglicht eine Termin- und Aufgaben-Verwaltung innerhalb von Vim.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/itchyny/calendar.vim

Zusätzlich habe ich in meiner Konfigurationsdatei folgende Einstellungen für das Calendar-Plugin vorgenommen:

```
" Calendar
    let g:calendar_google_calendar = 0
    let g:calendar_google_task = 0
    let g:calendar_cache_directory= expand("~/data/calendar/")
    let g:calendar_week_number=1
    let g:calendar_task_delete=1
    let g:calendar_task=0
    nmap gC :Calendar<CR>
```

In Vim kann das Calendar-Plugin damit über die Tastenkombination `g C` gestartet werden.

Eregex

Das Eregex-Plugin ermöglicht es, in Vim für das Suchen und Ersetzen Perl-kompatible reguläre Ausdrücke zu verwenden.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/othree/eregex.vim

Zusätzlich sollte die Konfigurationsdatei `~/.vimrc` um folgende Einträge ergänzt werden:

```
let g:eregex_default_enable = 1
let g:eregex_forward_delim = '/'
let g:eregex_backward_delim = '?'
nmap <leader>/ :call eregex#toggle()<CR>
```

Gibt man anschließend in einer neuen Vim-Sitzung `/` oder `?` ein, so erscheint in der Kommandozeile automatisch `:1M/` oder `:1M?`. Die unmittelbar dahinter eingegebenen Zeichen werden als Perl-compatible reguläre Ausdrücke (PCRE) interpretiert. Durch Eingabe von `\/` im Normalmodus kann dieses Verhalten aus- beziehungsweise wieder angeschaltet werden.

Um Ersetzungen mit PCRE-Syntax vorzunehmen, kann man im Kommandozeilen-Modus `S` statt `s` verwenden:

```
:[Bereich]S/PCRE-Syntax/Ersetzung/[Optionen]
```

Ebenso kann in `global`-Anweisungen die PCRE-Syntax verwendet werden, wenn diese mit statt `G` statt mit `g` eingeleitet werden:

```
:[Bereich]G/PCRE-Syntax/[Anweisung]
```

Für das Schreiben von Vim-Scripts bietet das Plugin auch eine Hilfe: Schreibt man in der aktuellen Datei einen regulären Ausdruck in Perl-Syntax und markiert ihn visuell, so kann dieser mittels `:E2v` in einen regulären Ausdruck mit Vim-Syntax übersetzt werden.

Beispiel:

```
# Entfernen von Tabs und Leerzeichen am Zeilenende:
# Perl-Syntax:
:%s/\s+$//g

# Visuell markieren, :E2v eingeben (wird ergänzt zu :'<,'>E2v)
# Ergebnis:
:%s/\s\+$//g
```

Eine gute Übersicht über reguläre Ausdrücke in Perl-Syntax findet sich beispielsweise hier.

Minibuf-Explorer

Das Minibuf-Explorer-Plugin bietet in einem eigenen kleinen Subfenster am unteren Fensterrand eine Übersicht über die aktuell geöffneten Buffer. Angezeigt werden standardmäßig die Nummern und eine abgekürzte Bezeichnung der Buffer. Um einen bestimmten Buffer auszuwählen, kann man in diesem Fenster mit `h j k l` den gewünschten Buffer anwählen und `Enter` drücken. Alternativ dazu kann beispielsweise `:b5` zur Auswahl des fünften Buffers oder `:bp` bzw. `:bn` zur Auswahl des vorherigen bzw. nächsten Buffers eingegeben werden, da die Buffer-Nummern ja stets angezeigt werden.

Durch folgende Zeilen in der `~/.vimrc` kann das Plugin so konfiguriert werden, dass die Bufferleiste stets unten am Bildschirm angezeigt wird und mit `F4` an- und ausgeschaltet werden kann:

```
let g:miniBufExplSplitBelow=1
map <F4> :MBEToggle<CR>
hi MBEVisibleActiveNormal guifg=magenta ctermfg=magenta
hi MBEVisibleActiveChanged guifg=magenta ctermfg=magenta
```

Durch die `hi`-Angaben wird der aktive Buffer in der Liste durch die Farbe `magenta` hervorgehoben.

Das Plugin ist auch in Verbindung mit der Option `swapfile` sinnvoll, die verhindert, dass eine Datei mehrfach geöffnet wird. Bei Verwendung des Minibuf-Explorers sieht man an jedem Vim-Fenster sofort, welche Dateien dort geöffnet sind.

Nerd-Commenter

Das NerdCommenter-Plugin ermöglicht es einzelne Zeilen oder (in Verbindung mit visuellen Markierungen) ganze Code-Abschnitte auszukommentieren. Dabei wird automatisch für jeden Filetype das passende Kommentarzeichen gewählt.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/scrooloose/nerdcommenter

Für die Benutzung des NerdCommenter-Plugins gibt es unter anderem folgende Tastenkombinationen:

`\cc`	Kommentiere (visuell) markierte Zeilen aus	(*comment*)
`\cu`	Kommentiere (visuell) markierte Zeilen ein	(*undo-comment*)
`\c Leertaste`	Kommentiere wechselhaft ein oder aus (kann häufig `\cc` und `\cu` ersetzen)	
`\cs`	„schickes“ Auskommentieren von langen Abschnitten (z.B. in C)	
`\cl`	Auskommentieren mit linksbündigen Kommentarzeichen (z.B. in RST oder Python)	

In der Datei `~/.vim/bundle/nerdcommenter/plugin/NERD_commenter.vim` können Kommentarzeichen für die verschiedenen Dateitypen einfach angepasst und/oder ergänzt werden. Dazu sucht man mit der Vim-Suche nach der gewünschten Endung, beispielsweise `tex`, und gibt wie bei den übrigen Einträgen das gewünschte Kommentarzeichen an.

Renamer

Das Renamer Plugin ermöglicht ein gleichzeitiges, fein steuerbares Umbenennen mehrerer Dateien mittels Vim.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/qpkorr/vim-renamer

In einem neuen Vim-Buffer kann mittels `:Ren` der Inhalt des aktuellen Verzeichnisses eingelesen werden. In der so erstellten Liste ist das Suchen und Ersetzen von Text (inklusive regulärer Ausdrücke) wie üblich möglich; mit `Enter` kann zudem in das Verzeichnis unter dem Cursor gewechselt werden.

Um die Dateien eines Verzeichnisses unmittelbar aus der Shell heraus mit Vim umzubenennen, kann `vim -c Ren` aufgerufen werden; hierfür kann wiederum in der `~/.bashrc` ein alias definiert werden, beispielsweise `alias vren='vim -c Ren'`.

Beim Umbenennen ist lediglich zu beachten, dass die Reihenfolge der Dateien nicht geändert werden darf und die Liste nach dem Umbenennen genauso viel Zeilen beinhalten muss wie zu Beginn (da jede Zeile genau einen Dateinamen beinhaltet).

Ist man mit dem Umbenennen fertig, gibt man nochmals `:Ren` ein, und die Dateien im jeweiligen Verzeichnis werden entsprechend umbenannt. :-)

Sudo

Das Sudo Plugin ermöglicht es, sich auch nachträglich mit SuperUser-Rechten ausstatten. Nützlich ist das, wenn man Systemdateien verändert, und es einem erst beim Speichern auffällt, dass man eigentlich gar keine Schreibrechte besitzt.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/vim-scripts/sudo.vim

Zur Verwendung des Sudo-Plugins gibt es folgende Funktionen für die Vim-Kommandozeile:

`:SudoWrite Datei`	Speichere `Datei` mit Root-Rechten (`:SudoWrite %` speichert so die aktuelle Datei ab)
`:SudoRead Datei`	Lese `Datei` mit Root-Rechten
`:e sudo:/path/Datei`	Öffne `Datei` mit Root-Rechten

Praktisch ist auch eine Abkürzung in der Konfigurationsdatei `~/.vimrc`:

```
cabbrev sw SudoWrite%            " Aktuellen Buffer mit Sudo-Rechten schreiben
```

SuperTab

Das SuperTab-Plugin bietet eine einfach Möglichkeit, im Einfüge-Modus mittels `Tab` das bis zum Cursor reichende Wort zu vervollständigen (ähnlich wie durch Verwendung von `Control x`).

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/ervandew/supertab

SuperTab bietet eine Möglichkeit, die Vervollständigung auf den Kontext bezogen durchzuführen. Gibt man beispielsweise einen Pfadnamen ein, so versucht SuperTab diesen zu vervollständigen; schlägt dies fehl, so wird versucht eine Vervollständigung anhand des bisher in dem aktuell geöffneten (oder weiteren) geöffneten Buffern zu bewirken. Hierzu muss folgender Eintrag in die `~/.vimrc` aufgenommen werden:

```
let g:SuperTabDefaultCompletionType = "context"
```

Gibt es mehrere Möglichkeiten zur Vervollständigung, so wird ein kleines Popup-Fenster, wobei die einzelnen Möglichkeiten mit `Control n`, `Control p` oder wiederum mit `Tab`

durchlaufen werden können. Der aktuelle Vervollständigungsvorschlag wird von SuperTab automatisch eingeblendet; drückt man die Leertaste oder fährt man fort zu schreiben, so wird der Vorschlag übernommen.

Hinweis: Bei Verwendung von SuperTab bewirkt die Tab-Taste nur noch ein Einfügen eines Tabulator-Zeichens als Abstandmarker, wenn dieses am Zeilenanfang steht oder wenn ein vor dem Cursor (mindestens) ein Leerzeichen steht; andernfalls wird durch die SuperTab-Funktion das Wort vor der aktuellen Position ergänzt.

Table-Mode

Das Table-Mode-Plugin ermöglicht es, mit dem Vim auf einfache Weise RestructuredText-Tabellen zu schreiben.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/dhruvasagar/vim-table-mode

Nach der Installation kann das Plugin in einer RestructuredText-Datei (Endung `.rst`) mittels `\tm` („table-mode-toggle“) aktiviert beziehungsweise wieder deaktiviert werden. Ist das Plugin aktiv, so werden eingegebene Textzeilen, die mit `|`-Zeichen beginnen, als Tabellen gewertet. Beispielsweise kann dann folgende Zeile als „Kopfzeile“ einer Tabelle eingegeben werden:

```
| Einträge | Eigenschaft 1 | Eigenschaft 2 |
```

Gibt man in der folgenden Zeile `||` (ein doppeltes Pipe-Zeichen) ein, so wird dieses automatisch wie folgt ergänzt:

```
| Einträge | Eigenschaft 1 | Eigenschaft 2 |
+----------+---------------+---------------+
```

Der entstandene „Querstrick“ kann entweder im Normalmodus mittels `yy` kopiert und über der Überschrift wieder eingefügt werden; alternativ dazu kann auch oberhalb der Überschrift `||` eingegeben werden, um eine Vervollständigung durch das Table-Mode-Plugin zu erreichen.

```
+----------+---------------+---------------+
| Einträge | Eigenschaft 1 | Eigenschaft 2 |
+----------+---------------+---------------+
```

Wird in weiteren, mit `|` beginnenden Zeilen Text einzugeben, so erfolgt bei Bedarf automatisch eine Verbreiterung/Anpassung der Spalten; durch jedes Drücken von `|` wird das aktuell eingegebene Feld automatisch auf die jeweilige Spaltenbreite angepasst und zur nächsten Spalte gewechselt.

Wird zu einem späteren Zeitpunkt ein Eintrag geändert, so kann mittels `\tre` („table-mode-refresh“) eine Aktualisierung der gesamten Tabelle erreicht werden.

Tagbar

Das Tagbar-Plugin bietet eine Art Inhaltsverzeichnis für Quellcode. Es nutzt das externe Programm `exuberant ctags`, um aus den aktuell geöffneten Dateien eine Übersicht an Funktionsnamen, Makros, Variablen, Klassen, usw. zu erstellen. In LaTeX -Dokumenten wird eine Kapitel-, Tabellen- und Labelübersicht angezeigt. Faltungen und Suchanweisungen funktionieren wie gewohnt.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/majutsushi/tagbar

Mittels `:TagbarToggle` oder einem entsprechenden Mapping in der *Konfigurationsdatei* wird rechts ein Fenster mit der Tagliste eingeblendet. Drückt man im Taglisten-Fenster über einem Schlagwort `Enter`, so wird im Hauptfenster das entsprechende Dokument an der jeweiligen Position geöffnet. Möchte man das Tag-Fenster auf der linken statt auf der rechten Seite platziert haben, so kann man die `~/.vimrc` um folgende Zeile ergänzen:

```
let g:tagbar_left = 1

"Optional: Tagbar mit F2 aufrufen:
nmap <F2> :TagbarToggle<CR>
```

Tagbar kann auch verwendet werden, um ein Inhaltsverzeichnis von *RestructuredText*-Dateien anzuzeigen. Hierzu muss ein kleines Python-Skript namens rst2ctags installiert werden. Ich persönlich habe mich bei der Installation für mein `~/.vim`-Verzeichnis als Zielpfad entschieden:

```
cd ~/.vim

# rst2ctags in das Verzeichnis ~/.vim/rst2ctags installieren:
git clone https://github.com/jszakmeister/rst2ctags
```

Anschließend muss gemäß dieser Anleitung noch folgende Ergänzung in der Konfigurationsdatei `~/.vimrc` vorgenommen werden:

```
let g:tagbar_type_rst = {
    \ 'ctagstype': 'rst',
    \ 'ctagsbin' : '~/.vim/rst2ctags/rst2ctags.py',
    \ 'ctagsargs' : '-f - --sort=yes',
    \ 'kinds' : [
        \ 's:sections',
        \ 'i:images'
    \ ],
    \ 'sro' : '|',
    \ 'kind2scope' : {
        \ 's' : 'section',
    \ },
```

```
    \ 'sort': 0,
\ }
```

Öffnet man nun in einer neuen Vim-Sitzung eine `rst`-Datei, so wird bei Aktivierung der Tagbar automatisch ein Inhaltsverzeichnis der aktuellen Datei eingeblendet. Wechselt man (beispielsweise mit `Ctrl w h`) in das Tagbar-Fenster, so kann man mit den normalen Navigationsbefehlen ein Kapitel auswählen und gelangt durch ein Drücken von `Enter` auf die entsprechende Kapitel-Zeile im Hauptfenster.

Ultisnips

Das Ultisnips-Plugin ist eine Weiterentwicklung des Snipmate-Plugins mit erheblich größerem Funtkionsumfang. Das Plugin ermöglicht es durch Eingabe kurzer, selbst definierter Textstücke („Snippets“) und Drücken der Tab-Taste diese durch entsprechende Templates zu ersetzen.

Mittels *Vundle* kann dieses Plugin über folgendes Repository installiert werden:
https://github.com/sirver/ultisnips

Vordefinierte Beispielsnippets finden sich im Paket vim-snippets, das zusätzlich installiert werden sollte. Bei der Verwendung von Vundle empfiehlt sich dabei die *pinned*-Option zu verwenden, damit eigene Änderungen in den Snippets-Dateien nicht durch Aktualisierungen überschrieben werden. Es sollten also folgende beiden Zeilen im Plugins-Abschnitt der Konfigurationsdatei `~/.vimrc` stehen:

```
Plugin 'sirver/ultisnips'
Plugin 'honza/vim-snippets', {'pinned': 1}
```

Nach der Installation der beiden Plugins befinden sich die zu den einzelnen Filetypes gehörenden Snippets im Verzeichnis `~/.vim/bundle/vim-snippets/UltiSnips/`; beispielsweise beinhaltet die Datei `python.snippets` in diesem Verzeichnis alle Snippets, die für Python-Dateien relevant sind. Die Snippets in der Datei `all.snippets` gelten für alle Dateitypen gleichermaßen.

Zur Verwendung des Ultisnips-Plugin habe ich zudem folgende Zeilen in die Konfigurationsdatei `~/.vimrc` aufgenommen:

```
" Snippets mit Tab vervollständigen, mit S-Tab mögliche Snippets anzeigen:
let g:UltiSnipsExpandTrigger="<tab>"
let g:UltiSnipsListSnippets="<s-tab>"

" Mit C-h und C-l zur vorherigen bzw. nächsten Snippet-Position springen:
let g:UltiSnipsJumpForwardTrigger="<c-l>"
let g:UltiSnipsJumpBackwardTrigger="<c-h>"

" Weitere Einstellungen:
```

```
let g:UltiSnipsSnippetsDir="~/.vim/bundle/vim-snippets/UltiSnips"
let g:UltiSnipsEditSplit="horizontal"
let g:UltiSnipsEnableSnipMate=0
```

Einzelne Snippets haben folgende Syntax:

```
snippet shortkey "Beschreibung" optionen
... template ...
endsnippet
```

Beispiel: (Definiert in `~/.vim/bundle/vim-snippets/UltiSnips/tex.snippets`)

```
snippet / "Math Fraction" w
\frac{$1}{$2}$0
endsnippet
```

Wird mit dieser Snippet-Definition in einer `.tex`-Datei im Einfügemodus das Zeichen `/` eingegeben und `Tab` gedrückt, so wird dieses Zeichen durch `\frac{}{}` ersetzt und der Cursor an die mit `$1` bezeichnete Stelle bewegt. Durch ein Drücken der Jump-Forward-Taste, die bei der obigen Konfiguration mit `<C-l>` definiert ist, gelangt man zur zweiten Sprungmarke `$2`; durch Drücken der Jump-Backwards-Taste, die mit `<C-h>` definiert ist, kann man umgekehrt wieder zur vorherigen Sprungmarke zurückkehren. Erreicht man schließlich, gegebenenfalls durch mehrmaliges Drücken der Jump-Forward-Taste, die Position `$0`, so wird das Ergänzen des Snippets abgeschlossen; die vorherigen Sprungmarken können dann nicht mehr angesteuert werden.

Snippets-Beispieldateien:

- `LaTeX-Snippets`
- `LaTeX-Math-Snippets`
- `LaTeX-Template-Snippets`
- `RestructuredText-Snippets`

Vorgabewerte und Snippets für visuell markierte Bereiche

Bei der Definition von Snippets können die Sprungmarken auch als `${1}`, `${2}` usw. angegeben werden. Dies nutzt man insbesondere dann, wenn man an den Sprungstellen mittels `${1:Vorgabe}` einen Standard-Text einfügen mag, der bei der Ergänzung des Snippets an dieser Stelle eingefügt wird. Gelangt der Cursor durch Drücken der Jump-Forward-Taste zu so einer Position mit Textvorgabe, so kann diese durch ein erneutes Drücken der Jump-Forward-Taste bestätigt werden; gibt man hingegen einen beliebigen anderen Text ein, so wird die Textvorgabe durch diesen ersetzt. Beispielsweise wird mittels `${2:$1}` der bei `$1` eingegebene Text automatisch als Vorgabewert an der Stelle `$2` eingefügt.

Ein besonderer Vorgabewert ist `${VISUAL}`: Diese Variable enthält den visuell markierten Textbereich, wenn vom visuellen Modus ausgehend `Tab` gedrückt wird. Man kann sich damit Snippets definieren, die wahlweise auf visuell markierte Textbereiche angewendet werden können oder andernfalls ein leeres Template erzeugen:

```
snippet cen "Centered Text" b
\begin{center}
    ${1:${VISUAL:}}
\end{center}
$0
endsnippet
```

Das obige Snippet kann auf zweierlei Arten verwendet werden:

- Im Einfügemodus wird durch Eingabe von `cen<Tab>` eine `center`-Umgebung erzeugt und der Cursor an die Stelle `${1}` gesetzt; da `${VISUAL}` hierbei leer ist, wird an `${1}` kein Text eingefügt (es könnte auch `${VISUAL:Standard}` angegeben werden, um einen Vorgabewert zu setzen, wenn `${VISUAL}` leer ist).
- Im visuellen Modus kann ein Textbereich markiert und `<Tab>cen<Tab>` eingegeben werden. Dabei verschwindet während der Eingabe von `cen` der visuell markierte Bereich; drückt man wieder `<Tab>`, so wird er als Vorgabewert für `${1}` wieder eingeblendet. Drückt man die Jump-Forward-Taste, so wird dieser Vorgabewert übernommen und man gelangt an das Ende des Snippets (`$0`).

Derartige Snippets können, ähnlich wie das Surround-Plugin, Textbereiche in gewünschte Umgebungen setzen.

Snippet-Optionen und reguläre Ausdrücke

Durch die Angabe von Optionen kann gezielter festgelegt werden, wann ein Snippet durch Drücken von `<Tab>` ausgelöst werden soll:

- Die Option `w` („word“) besagt, dass das Snippet nur dann ausgelöst wird, wenn das Kürzel ein eigenständiges Wort bildet, also unmittelbar vor dem Kürzel ein Whitespace-Zeichen (Leerzeichen, Tab, usw.) steht. Wird keine Option angegeben, wird automatisch `w` als Standard-Kriterium verwendet.

  Beispiel:

  Folgendes Snippet für `.rst`-Dateien fügt durch Drücken von `mi<Tab>` eine Math-Inline-Umgebung ein:

  ```
  snippet mi "Math Inline" w
  :math:\`$1\` $0
  ```

  Die Bedeutung der Backticks (``` `` ```) als Begrenzungszeichen für *Scripte innerhalb eines Snippets* muss im obigen Beispiel mit je einem Backslash (\) aufgehoben werden, um eine Interpretation des Inhalts zwischen den Backticks als Shell-Skript zu verhindern.

  Das Snippet soll nur ausgelöst werden, wenn `mi` nicht Teil eines Wortes ist; beispielsweise soll eine Expansion vermieden werden, wenn `vermi<Tab>` eingegeben wird (um beispielsweise dieses Wort mittels des *SuperTab-Plugins* zu „vermieden“ o.ä. zu ergänzen.)

- Die Option `b` („begin of line“) bewirkt, dass das Snippet nur dann ausgelöst wird, wenn das Kürzel am Anfang einer Zeile steht. Whitespace-Zeichen (Leerzeichen, Tab, usw.) am Beginn der Zeile werden dabei ignoriert.

  Beispiel:

  Folgendes Snippet für `.rst`-Dateien fügt durch Drücken von `ma<Tab>` eine Math-Paragraph-Umgebung ein:

  ```
  snippet ma "Math Paragraph" b
  .. math::

      ${1}

  $0
  endsnippet
  ```

  Das Snippet soll allerdings nicht ausgelöst werden, wenn `ma` Teil eines Wortes ist oder mitten in der Zeile vorkommt.

- Die Option `i` („inner word“) bewirkt, dass das Snippet auch dann ausgelöst wird, wenn es innerhalb eines Wortes vorkommt.

  Persönlich verwende ich derartige Snippets, um beispielsweise durch Eingabe von `a<Tab>` oder `ae<Tab>` den deutschsprachigen Umlaut `ä` zu erzeugen. Damit ist es ohne Mehraufwand möglich, auch bei Verwendung eines englischen Tastaturlayouts deutschsprachigen Text zu schreiben.

- Die Option `r` kann in Verbindung mit den Optionen `i`, `w`, und `b` angegeben werden, um zu bewirken, dass das Snippet-Kürzel als regulärer Ausdruck mit Python-Syntax interpretiert wird; Das Kürzel muss dabei in Anführungszeichen gesetzt werden.

  Beispiel:

  Die folgenden Snippets ermöglichen als Inner-Word-Snippets die Umwandlung von `a<Tab>`, `ae<Tab>` usw. in deutschsprachige Umlaute:

  ```
  snippet "ae?" "ä" ri
  ä$0
  endsnippet
  snippet "Ae?" "Ä" ri
  Ä$0
  endsnippet
  snippet "oe?" "ö" ri
  ö$0
  endsnippet
  snippet "Oe?" "Ö" ri
  Ö$0
  endsnippet
  snippet "ue?" "ü" ri
  ü$0
  endsnippet
  snippet "Ue?" "ü" ri
  ```

```
Ü$0
endsnippet
snippet "ss?" "ß" ri
ß$0
endsnippet
```

Das Zeichen `?` in der Snippet-Definition steht dabei für `0` oder `1` Vorkommen des vorherigen Zeichens.

Prioritäten

Gibt es zu einem im Einfügemodus eingegebenen Textstück mehrere mögliche Snippets, so werden diese beim Drücken von `Tab` nummeriert und unter Angabe der jeweiligen Snippet-Datei aufgelistet und können durch Eingabe von `1`, `2`, usw. ausgewählt werden. Üblicherweise wird allerdings eine eindeutige und somit schnelle Ergänzung der Snippets bevorzugt. Dies lässt sich in einer Snippets-Datei durch die Vergabe von Prioritäten mittels `priority num` erreichen, wobei `num` einen Wert zwischen `-50` und `+50` bezeichnet. Alle Snippets, die unterhalb einer solchen Eingabezeile stehen, bekommen diese Priorität zugewiesen (bis zum Ende der Datei oder bis zur nächsten `priority`-Zeile).

Die vordefinierten Snippets aus dem `vim-snippets`-Plugin haben alle als Priorität `-50`; sie werden also nur dann ausgeführt, wenn kein anderes (auch gleichnamiges) Plugin mit höherer Priorität existiert.

Beispielsweise haben bei mir die Umlaut-Snippets die Priorität `-10`, so dass sie nur dann ausgeführt werden, wenn kein anderes Snippet auf den eingegebenen Text zutrifft; beispielsweise soll gemäß des obigen Beispiels `ma<Tab>` am Anfang einer Zeile zu einer `math`-Umgebung expandiert werden, innerhalb einer Zeile soll `ma<Tab>` hingegen zu `mä` expandiert werden, wenn beispielsweise „mäkeln“ geschrieben werden soll.

Ohne die explizite Angabe einer Priorität haben Snippets (beispielsweise in einer neuen Snippet-Datei) die Priorität Null. Man kann sich damit zusätzliche Snippets in eigenen Dateien definieren, beispielsweise `textemplates.snippets` für eigene LaTeX-Dokumentvorlagen oder `texmath.snippets` für Mathe-Snippets. Man kann eine Snippets-Datei mittels des Schlüsselwortes `extends` um weitere „Dateitypen“ erweitern:

```
# Innerhalb der Datei ``tex.snippets``:
# Zusätzlich die Snippets in folgenden Dateien (ohne Dateiendung)␣
↪berücksichtigen:

extends textemplates, texmath
```

Beispielsweise können so LaTeX-Mathe-Snippets zentral sowohl für `.tex` wie auch für `.rst`-Dateien definiert werden. Das spart nochmals Schreibarbeit – don’t repeat yourself!

Einfache Ersetzungen

Soll der an der Stelle `$1` eingegebene Text auch an einer Stelle erscheinen, so gibt man dort erneut `$1` oder beispielsweise `${2:$1}` ein, sofern `$1` nur ein Vorgabewert sein soll. Man kann bei der erneuten Verwendung von `$1` den dort gespeicherten Inhalt allerdings auch abändern, indem man `${1/search/replace/}` eingibt.

Beispielsweise werden in den Grund-Wissen-Tutorials oft `.png`-Bilder eingefügt und dabei in der Fußzeile die zugehörigen `.svg`-Vektorgraphiken mit als Download-Option verlinkt. Um dabei den Dateinamen nur einmal eingeben zu müssen, kann folgendes Snippet verwendet werden:

```
snippet figs "Figure with SVG" b
.. figure:: ${1:path}
    :name:  fig-${2}
    :alt:   fig-${3:$2}
    :align: center
    :width: 50%

    .. only:: html

        :download:\`SVG: ${1/png/svg/}>\`

$0
endsnippet
```

Bei der Expansion dieses Snippets gelangt man zunächst an die Stelle `$1`, an der offensichtlich eine Pfadangabe erwartet wird. Gibt man hier einen Pfad ein, der mit `.png` endet, so erscheint in der Fußzeile automatisch der gleiche Pfad mit der Endung `.svg`; möchte man diesen so automatisch generierten Pfad nur als Vorgabewert haben, kann man an dieser Stelle auch `${4:${1/png/svg/}}` schreiben, um eine entsprechende zusätzliche Sprungmarke zu definieren.

Ausführen von Scripten

Weitere Möglichkeiten für Snippets bieten sich dadurch, dass innerhalb von Snippets wahlweise Vim-, Shell- oder Python-Scripts ausgeführt werden können. Diese können mehrere Zeilen umfassen und werden innerhalb der Snippet-Templates folgendermaßen begrenzt:

- `` ` ... ` ``: Shell-Script
- `` `!v ... ` ``: Vim-Script
- `` `!p ... ` ``: Python-Script

Verwendet man Python-Scripts, so werden automatisch die Module `vim`, `re`, `os`, `string` und `random` geladen; zudem sind automatisch folgende Variablen vordefiniert:

`snip`	Ein zum aktuellen Snippet gehörendes Snippet-Objekt
`fn`	aktueller Dateiname
`path`	Absoluter Pfad der aktuellen Datei
`t`	Liste mit den Inhalten von `$1`, `$2`, usw. (`t[1]` entspricht dem Inhalt von `$1` usw.)

Ein `snip`-Objekt hat dabei unter anderem folgende Attribute:

- In der Variable `snip.rv` wird der Rückgabewert des Snippets gespeichert.
- Die Variable `snip.basename` enthält den Namen der aktuellen Datei ohne Dateiendung.
- Die Variable `snip.ft` enthält den Namen des aktuellen Filetypes.
- Die Variable `snip.v` enthält Daten, die sich auf `${VISUAL}` beziehen: `snip.v.text` gibt den Inhalt von `${VISUAL}` an, `snip.v.mode` hat als Wert entweder `v`, `V` oder `^V` je nach Art des visuellen Modus.

Durch Scripte bieten sich in Snippets nahezu unbegrenzte Möglichkeiten; man kann sich so geradezu temporäre „Buttons“ definieren, die bei der Expansion bestimmte Funktionen auslösen; beispielsweise können reguläre Ausdrücke in der Definition von Snippets gezielt ausgewertet werden:

```
# Snippet zum Ergänzen von "bei<Tab>" zu "beispielsweise":

snippet "(B|b)ei" "beispielsweise" rw
`!p
if match.group(1).islower():
    snip.rv = "beispielsweise"
else:
    snip.rv = "Beispielsweise"
` $0
endsnippet
```

Hierbei steht `match.group(1)` für den konkreten Wert, der sich bei der Auswertung der ersten in der Snippet-Definition auftretenden Gruppierung `(b|B)` ergibt.

Weitere, auch umfangreichere Beispiele zum Einsatz von Scripten sind in den Snippet-Dateien von des `vim-snippets`-Plugins enthalten.

Vicle

Das Vicle Plugin ermöglicht es, von Vim aus mittels `Ctrl c Ctrl c` die aktuelle Zeile oder im visuellen Modus ganze Codeblöcke an eine offene *Screen*- oder *tmux*-Sitzung zu schicken. Egal ob Python, R, MySQL oder die Shell selbst als Interpreter verwendet wird: Skript-Teile lassen sich auf diese Weise bereits während des Erstellens „on-the-fly“ testen.

Mittels *Vundle* kann dieses Plugin über folgendes Github-Repository installiert werden: https://github.com/vim-scripts/Vicle

Um ein `tmux`-Fenster als Ziel für den übergebenen Code zu verwenden, muss man die `~/.vimrc` um folgenden Eintrag ergänzen:

```
let g:vicle_use = 'tmux'
```

Drückt man in einer Vim-Sitzung erstmals `Ctrl c Ctrl c`, so wird man aufgefordert, wahlweise den Namen oder die Nummer der `tmux`-Session und des Zielfensters einzugeben; hat das Zielfenster mehrere Teilfenster („panes“), so kann beispielsweise `2.3` das dritte Pane im zweiten Fenster bezeichnet werden. In `tmux` werden die Panes allgemein in der gleichen Reihenfolge nummeriert, wie sie geöffnet wurden; mittels `<tmux-hotkey>:list-panes` können die Pane-Nummern eines Fensters explizit angezeigt werden.

Im folgenden schickt Vicle bei einer Eingabe von `Ctrl c Ctrl c` von dieser Vim-Sitzung aus stets die aktuelle oder die visuell markierten Zeilen an das angegebene Zielfenster. Von mehreren verschiedenen Vim-Sitzungen aus kann Text somit an verschiedene (oder auch an den gleichen) Interpreter geschickt werden.

Yankring

Das Yankring Plugin speichert automatisch der Reihenfolge nach die zuletzt in die interne Zwischenablage kopierten Inhalte, so dass sie gezielt an einer anderen Stelle und/oder zu einem späteren Zeitpunkt wieder eingefügt werden können.

Mittels *Vundle* kann dieses Plugin über folgendes Github-Repository installiert werden: https://github.com/vim-scripts/YankRing.vim

Wird der Inhalt der Vim-internen Zwischenablage im Normalmodus mit `p` oder `P` eingefügt, kann mit `Ctrl p` und `Ctrl n` anstelle dessen der rückwärts bzw. vorwärts in der Kopier-History nächstgelegene Inhalt ausgewählt und eingefügt werden.

Links

- Die Vim-Projektseite

Auf folgenden Seiten finden sich weitere Infos rund um Vim:

- Vim Survival Guide
- Selflinux Vim Tutorial
- A Byte of Vim (free online book)

- Rechtschreibprüfung mit Vim
- Superuser Vim Questions
- Vim Regular Expressions 1
- Vim Regular Expressions 2
- Scripting Vim

Vim-Lehrbuch:

- Learn Vim-Scripting (The Hard Way)

Vim-Kurzübersichten:

- Vim Cheat Sheet 1
- Vim Cheat Sheet 2 (PDF)

Der Email-Client `mutt`

Mutt ist ein textbasierter Email-Client, der sich durch eine hohe Funktionalität und Konfigurierbarkeit auszeichnet. Dadurch, dass es sich um eine textbasierte Shell-Anwendung handelt, können HTML-Emails nur auf minimalistische Weise (mittels lynx) dargestellt werden. Dafür kann die Verwaltung der Emails erfolgt ausschließlich über die Tastatur erfolgen, was eine sehr schnelle Bedienung ermöglicht.

- Mutt kann die einkommenden Emails in Thread-Darstellung anzeigen und in der Übersicht farblich nach benutzerdefinierten Kriterien hervorheben.
- Mutt unterstützt zudem die Nutzung unterschiedlicher Mail-Boxen, so dass Emails beispielsweise automatisch in Jahres-Archiven abgelegt werden können.
- Mutt bietet Funktionen zum sehr schnellen Durchsuchen von Mail-Boxen nach Inhalten (wahlweise in den eigentlichen Email-Inhalten, oder nur in den Betreffszeilen); es können auch Emails gemäß bestimmter Kriterien ausgewählt und daraufhin gemeinsam verschoben, gelöscht, oder kopiert werden.
- Zum Schreiben von Emails kann zudem jeder beliebige Text-Editor verwendet werden (beispielsweise *Vim*).

Installation von Mutt und Hilfsprogrammen

Zur Installation von `mutt` sollten folgende Pakete installiert werden:

```
sudo aptitude install fetchmail procmail mutt lynx msmtp
```

Das Hilfsprogramm `fetchmail` holt Emails von einem Email-Provider ab, das Programm `procmail` leitet sie (gegebenenfalls mit selbst definierten Spam-Filtern) an eine oder mehrere Mailboxen weiter. Mit `mutt` werden die Emails und Mailboxen verwaltet, `lynx` dient

zum Betrachten von HTML-Emails und `msmtp` als Hilfsprogramm zum Verschicken eigener Emails.

Vor dem ersten Start sollte zunächst im Home-Verzeichnis `~` ein `Mail`- und ein `.tmp`-Ordner für Mailboxen und temporäre Dateien angelegt werden:

```
mkdir ~/Mail ~/.tmp
```

Anpassen der Konfigurationsdateien

Anschließend sollten die Konfigurationsdateien der einzelnen Programme angelegt und mit grundlegenden Inhalten gefüllt werden:

- In der Datei `~/.fetchmailrc` werden die zum Abholen der Emails nötigen Informationen abgelegt:

  ```
  # Datei .fetchmailrc

  poll EMAIL-HOST protocol pop3 user "EMAIL@ADRESSE.DE" password "GEHEIM" ssl
  ```

  Der Email-Host ist beispielsweise `pop.gmx.net`. Als Protokoll kann anstelle von `pop3` auch `imap` verwendet werden (mit passendem Host, beispielsweise `imap.gmx.net`). In jeder Zeile der Konfigurationsdatei kann ein neuer Eintrag stehen, so dass ein zentrales Abholen der Emails von mehreren Konten leicht möglich ist.

  Da in der Datei `.fetchmailrc` das Email-Passwort im Klartext enthalten ist, darf die Datei nur Lese- und Schreibrechte für den Eigentümer haben. Dazu gibt man folgendes ein:

  ```
  chown 600 ~/.fetchmailrc
  ```

  Ebenso ist es möglich, die Datei auf einer verschlüsselten Partition oder einem verschlüsselten USB-Stick abzulegen und im Home-Verzeichnis einen Symlink dorthin zu erstellen.

- In der Datei `~/.msmtprc` werden die zum Versenden der Emails nötigen Informationen angegeben:

  ```
  # Datei .msmtprc

  defaults
  auth            on
  tls             on

  account default
  host EMAIL-HOST
  port 587
  from EMAIL@ADRESSE.de
  user EMAIL@ADRESSE.de
  password GEHEIM
  ```

```
tls_trust_file /etc/ssl/certs/ca-certificates.crt
auth login
```

Der Email-Host ist beispielsweise `mail.gmx.net`. Inzwischen verlangen viele Email-Provider eine SSL- oder TLS-Verschlüsselung für die Verbindung zwischen dem Host und dem Client; dies wird durch die obigen Konfigurationen als Standard festgelegt, wobei der genutzte Port bei verschiedenen Providern unterschiedlich sein kann (Infos hierzu sollten von jedem Provider angegeben sein). Für verschiedene Email-Konten können wiederum verschiedene Benutzer-Accounts angelegt werden. Der jeweilils passende Account wird beim Versenden einer Email automatisch anhand des `From:`-Eintrags der Email ausgewählt.

Auch in dieser Datei ist das Passwort im Klartext (allerdings ohne Anführungszeichen) enthalten. Auch diese Datei darf somit nur Lese- und Schreibrechte für den Eigentümer haben:

```
chown 600 ~/.msmtprc
```

- In der Datei `~/.procmailrc` sind die zum Verteilen („processing“) der Emails nötigen Informationen gespeichert:

```
# Datei .procmailrc

MAILDIR=$HOME/Mail/
LOGFILE=$HOME/.procmaillog
LOGABSTRACT=no
VERBOSE=off
FORMAIL=/usr/bin/formail
NL=''

# Doppelt gesendete Mails mittels formail abfangen
:0 Whc: .msgid.  lock
| $FORMAIL -D 16384 .msgid.cache
:0 a
$MAILDIR/duplicates


# Spam nach Absender aussortieren
:0
* ^Sender:.*(video|price|addme)
$MAILDIR/spam

#Spam nach Betreff aussortieren
:0
* ^Subject:.*(credit|cheap|cash|money|debt|sale|loan)
$MAILDIR/spam


# Alle anderen Emails in die default-Mailbox ablegen:
:0
```

```
* .*
default
```

Durch `:0` wird jeweils eine Filterregel eingeleitet. Anschliessend wird die eingehende Email auf ein Muster geprüft; beispielsweise würde `* ^From: .*Max Mustermann` bedeuten, dass für alle Emails mit „Max Mustermann“ im Absender die darauf folgende Aktions-Zeile ausgeführt wird.[1]

- In der Datei `~/.muttrc` wird das Verhalten von Mutt über eine Vielzahl möglicher Konfigurationen festgelegt. Dabei können Pfade, Tastenbelegungen, Farben, Verschlüsselungs-Einstellungen usw. angepasst werden. Die Datei kann beispielsweise so aussehen:

```
# Datei .muttrc

# ---------------------------------------------------------------------
# PFADEINSTELLUNGEN
# ---------------------------------------------------------------------

# Pfad für Adressbuch-Datei festlegen:
set alias_file=     "~/.mutt/addressbook"
source              "~/.mutt/addressbook"

# Standard-Mailbox für eingehende Emails:
set spoolfile='+default'

# Gelesene Emails nach "inbox-JAHR" im Mail-Ordner verschieben
# (beispielsweise "inbox-2014" für Emails aus dem Jahr 2014)
set mbox="+inbox-`date +%Y`"

# Gesendete Emails nach "sent-JAHR" im Mail-Ordner verschieben
set record="+sent-`date +%Y`"

# Email-Entwürfe in der Mailbox "Entwuerfe" speichern:
set postponed="+Entwuerfe"

# Pfad für temporäre Dateien festlegen:
set tmpdir=~/.tmp

# ---------------------------------------------------------------------
# TASTENKOMBINATIONEN
# ---------------------------------------------------------------------

# Mails durch Drücken von "A" vom Provider abholen und dort löschen:
macro index,pager A "!fetchmail -m 'procmail -d %T'\r"

# Alternativ: Mails durch Drücken von "A" vom Provider abholen und
# dort belassen (keep and verbose):
# #macro index,pager A "!fetchmail -kv -m 'procmail -d %T'\r"
```

[1] Eine ausführliche Beschreibung von Filterregeln findet sich beispielsweise auf dieser Seite.

```
bind browser <Enter> view-file

# HTML-Mails durch Drücken von "l" mit lynx betrachten:
macro pager,attach l "<pipe-entry>lynx -stdin -force_html<enter>"

# Emails durch Drücken von "f" weiterleiten
bind index,pager f forward-message

# An alle Empfänger einer Email antworten
bind index,pager R group-reply


# ---------------------------------------------------------------------
# ALLGEMEINE EINSTELLUNGEN
# ---------------------------------------------------------------------

# Alle Header-Infos grundsätzlich ausblenden:
ignore *

# Folgende Header-Infos jedoch erlauben:
unignore      from: subject to cc mail-followup-to \
    date x-mailer x-url list-id

# Format für das Zitieren der Original-Mail in einer Antwort-Email
# ("Am DATUM schrieb ABSENDER:")
set attribution="* %n <%a> [%(%d.%m.%Y %H:%M)]:"

set nobeep                # Keine akustischen Signale bei neuen
                          # Nachrichten
set copy=yes              # Gesendete Emails immer speichern
set delete=yes            # Als gelöscht markierte Emails beim Beenden
                          # löschen (ohne Nachfrage)
set editor='vim'          # Oder ein anderer Edigor, z.B. 'gedit'
set fast_reply            # Beim Antworten auf eine Email sofort den
                          # Editor öffnen
set followup_to           # In Betreff und Email-Header "Follow up"-
                          # Markierungen setzen
set help=no               # Hilfe-Zeilen ausblenden
set include=ask-yes       # Nachfragen, ob Original-Mail in Antwort
                          # zitiert werden soll

set move=yes              # Gelesene Nachrichten in die obige mbox
                          # verschieben
set nosave_empty          # Keine leeren Email-Entwürfe speichern
set pager_index_lines=6   # Beim Lesen von Emails 6 Zeilen für Pager
                          # reservieren
set pager_stop            # Beim Lesen von Emails nicht in die nächste
                          # Email scrollen
set read_inc=25           # Fortschritts-Anzeige beim Lesen von
                          # Mailboxen einblenden
```

```
set reply_to              # Antwort-Emails automatisch erkennen

set reply_regexp="^((re([\[^-][0-9]+\]?)*|Re|aw|antwort|antw|wg):[ \t]*)+"

set reverse_alias         # Namen von Email-Absendern anhand
                          # Adress-Liste anzeigen

set send_charset="us-ascii:iso-8859-1:iso-8859-15:iso-8859-2:utf-8"

set smart_wrap            # Besserer Zeilenumbruch
set sort=threads          # Emails nach in Thread-Reihenfolge anzeigen
set sort_aux=date-sent    # Emails innerhalb von Threads nach dem Datum
                          # sortieren
set strict_threads        # Bei Threading auf In-Reply-To-Header achten,
                          # nicht auf Betreff
set weed=yes              # Standard.. :)
set wrap_search=yes       # Im Index-Modus Suche erneut von vorne
                          # zulassen

auto_view text/html


# ---------------------------------------------------------------------
# PERSOENLICHE EINSTELLUNGEN
# ---------------------------------------------------------------------

my_hdr From:         "VORNAME NACHNAME" <EMAIL@ADRESSE.de>
my_hdr Reply-To:     "VORNAME NACHNAME" <EMAIL@ADRESSE.de>
set realname=        "VORNAME NACHNAME"
set signature=       "+.signature"

# Einstellungen für den Standard-Ordner:
folder-hook . "set from='VORNAME NACHNAME  <EMAIL@ADRESSE.de>'"
folder-hook . "set index_format='%4C %Z %{%b %d} %-15.15L (%4l) %s'"
folder-hook . "set sendmail='/usr/bin/msmtp --account=default'"


# ---------------------------------------------------------------------
# FARBEN
# ---------------------------------------------------------------------

# Aussehen von Mutt:

color     tree            brightmagenta     default
color     attachment      magenta           default
color     error           red               default
color     header          brightyellow      default   "^Subject: "
color     header          white             default   "^To:"
color     hdrdefault      yellow            default
color     indicator       black             white
color     markers         brightblue        default
```

```
color     message         white             default
color     normal          white             default
color     quoted          yellow            default
color     quoted1         green             default
color     quoted2         cyan              default
color     quoted3         red               default
color     signature       brightblack       default
color     status          brightyellow      blue
color     search          default             green

# Highlighting von Emails (abhängig von der "Punktezahl" einer Email):

# Mögliche Muster zur Punktevergabe:
# ~f ABSENDER     : Betrifft alle Emails, die vom ABSENDER
#                   geschickt wurden ("from")
# ~t EMPFAENGER   : Betrifft alle Emails, die an EMPFAENGER
#                   geschickt wurden ("to")
# ~s BETREFF      : Betrifft alle Emails, die BETREFF
#                   </EMAIL@ADRESSE>in der Betreff-Zeile enthalten
#                   ("subject")

# Reguläre Ausdrücke als Suchmuster:
# .       : Ein beliebiges Zeichen
# *       : Der vorherige Ausdruck Null mal oder beliebig oft
# [aA]    : Eines der in der Klammer enthaltenen Zeichen (a oder A)


# Alle Emails bekommen zunächst 0 Punkte:
unscore *

# Beispiel 1: 10 Punkte an alle Emails vergeben, die "grund-wissen"
# im Empfänger-Namen enthalten:
score '~t .*@grund-wissen.*' +10

# Beispiel 2: 25 Punkte an alle Emails vergeben, die "sphinx"
# in der Betreff-Zeile enthalten:
score '~s .*sphinx.*'      +25


# Zum Beispiel 1: Alle Emails mit einer Punktezahl von 10-20
# hellrot hervorheben:
color index brightred default '~n 10-20'

# Zum Beispiel 2: Alle Emails mit einer Punktezahl von 25-29
# blau hervorheben:
color index blue default      '~n 25-29'
```

In der obigen Beispiel-Konfigurationsdatei sollten die in Großbuchstaben gesetzten Variablen durch eigene Angaben ersetzt werden; zudem sollte der zum Schreiben von Emails bevorzugte Editor festgelegt werden.

Damit die Mailboxen im Verzeichnis `~/Mail` automatisch erkannt werden, sollte zudem folgender Eintrag zu den Pfadeinstellungen hinzugefügt werden:

```
# Mailboxen automatisch finden:
mailboxes $(find ~/Mail/ -maxdepth 0 -type d -printf "%p)
```

Dieser Eintrag befindet sich bei mir genau so in meiner eigenen `~/.muttrc`; leider wird jedoch in der Druckversion durch diese Zeile das Highlighting der gesamten Konfigurationsdatei deaktiviert.. anscheinend ein Fehler von Pygments, dem ansonsten echt prima funktionierenden Syntax-Highlighting-Tool.

Durch eine Vergabe von Punkten auf bestimmte Muster im Absenderfeld oder in der Betreffzeile von Emails ist es möglich, Emails von einzelnen Personen, Unternehmen oder Mailinglisten gezielt farblich hervorzuheben. Als Farben sind `red`, `green`, `blue`, `yellow`, `cyan`, `magenta` möglich, wobei durch die Nachrichten bei einem davor gestellten `bright` zusätzlich fett gedruckt erscheinen (beispielsweise steht `brightgreen` für grün und fettgedruckt).

Das Prinzip der Punktevergabe ist eigentlich simpel, man sollte lediglich darauf achten, dass die einkommenden Emails nicht mehrfach Punkte erhalten, beispielsweise weil sie das Wort „Python“ sowohl in der Emailadresse wie auch in der Betreffzeile enthalten. Sollten beide Muster beispielsweise mit 5 Punkten gewertet werden, so bekäme die Email insgesamt 10 Punkte und könnte dadurch gegebenenfalls eine andere Farbe bekommen..

Bedienung von Mutt

Startet man Mutt (durch Eingabe von `mutt` in einer Shell), so werden bei Verwendung der obigen Einstellungen die Emails der `~/Mail/default`-Mailbox aufgelistet. Dieser Ansichtsmodus von Mutt wird „Index“ genannt. Wird Mutt zum ersten Mal gestartet, ist die Index-Ansicht normalerweise leer.

Index-Modus

Im Index-Modus können folgende Funktionen durch Drücken der jeweiligen Taste aufgerufen werden:

- Email abrufen und zwischen Emails navigieren:

`A`	Emails vom Provider abrufen
`a`	Absender der Email unter dem Cursor ins Adressbuch aufnehmen
`?`	Hilfe einblenden
↓ oder `j`	Zur nächsten Email gehen
↑ oder `k`	Zur vorherigen Email gehen
`HOME`	Zur ersten Email gehen
`END`	Zur letzten Email gehen
`q`	Mutt beenden („quit“)

Zudem kann man mit `PageUP` und `PageDown` die Emails seitenweise durchblättern.

- Emails schreiben, beantworten und weiterleiten:

`m`	Neue Email verfassen („mail“)
`r`	Auf Email unter dem Cursor antworten (nur dem Absender) („reply“)
`R`	Auf Email unter dem Cursor antworten (allen Empfängern) („group reply“)
`f`	Email unter dem Cursor weiterleiten („forward“)

  Zum Schreiben der Emails wird automatisch der in der Konfigurationsdatei festgelegte Editor geöffnet. Speichert man dort die verfasste Email und beendet den Editor, kehrt man automatisch zu Mutt zurück.

- Emails markieren, löschen, verschieben, kopieren:

`d`	Email unter dem Cursor löschen („delete“)
`u`	Löschmarkierung unterhalb des Cursor aufheben („undelete“)
`t`	Email unter dem Cursor mit einer Markierung versehen („tag“)
`D`	Emails nach bestimmtem Muster löschen („delete by pattern“)
`U`	Löschmarkierungen nach bestimmtem Muster aufheben („undelete by pattern“)
`T`	Emails nach bestimmtem Muster mit einer Markierung versehen („tag by pattern“)
`w`	Status der Email-Adresse anpassen (`O`: Old, `N`: New, `D`: Delete, `r`: Responded , `*`: Tagged, `!` : Important)
`C`	Email unter dem Cursor in eine andere Mailbox kopieren („copy“)
`s`	Email unter dem Cursor in andere Mailbox abspeichern/verschieben („save“)

  Ist die Option `set move=yes` in der Konfigurationsdatei aktiviert, werden gelesene Emails automatisch beim Beenden von Mutt von der `default`-Mailbox in die `mbox`-Mailbox (bei obigen Einstellungen: `inbox-2014`) verschoben. Möchte man eine Email jedoch noch in der (meist deutlich kleineren) Mailbox `default` behalten, so kann man sie mittels `w o` wieder als ungelesen markieren.

  Um mehrere Emails auf einmal in eine andere Mailbox zu verschieben, markiert man mittels `t` zunächst die einzelnen Emails. Anschließend kann man mittels Eingabe von `;` den darauf folgenden Befehl auf alle markierten Emails anwenden; beispielsweise können durch `;s` alle markierten Emails in eine auszuwählende Mailbox verschoben werden.

- Emails durchsuchen:

`/`	Nach Emails suchen
`n`	Zur nächsten Email gehen, auf welche die Suche zutrifft
`N`	Rückwärts zur nächsten Email gehen, auf welche die Suche zutrifft

  Bei der Suche mittels `/Suchbegriff` werden nur die Email-Header, also insbesondere das Absenderfeld und die Betreffszeile durchsucht.

  Möchte man auch den Inhalt der Mails durchsuchen, kann man `/ ~b Suchbegriff` eingeben („search bodies“).

- Mit `c` kann man zwischen verschiedenen Mailboxen wechseln.

Bei mehreren der obigen Funktionen wird vom Benutzer eine weitere Eingabe von Text in der Eingabezeile (unten am Bildschirm) erwartet. Um diesen „Eingabe-Modus“ abzubrechen und wieder zum normalen Index zurückzukehren, muss man (etwas gewöhnungsbedürftig) `Ctrl g` drücken.

Pager-Modus

Drückt man im Index-Modus `Leertaste` oder `Enter`, so wird der Inhalt der Email im so genannten Pager-Fenster angezeigt. In diesem kann man mit den Pfeiltasten oder `PageUp` und `PageDown` durch den Inhalt der Email scrollen. Durch Drücken von `q` gelangt man zurück in den Index-Modus. Mittels `r` kann man die aktuelle Email unmittelbar beantworten oder mittels `d` löschen; Mutt zeigt dann automatisch die nächste Email im Pager an.

Anhänge von Emails können im Pager-Modus mittels `v` angezeigt werden. Mit den Cursor-Tasten kann dann ein Anhang ausgewählt und mittels `s` gespeichert werden. Mutt speichert den Anhang dabei in dem Verzeichnis, aus dem heraus Mutt aufgerufen wurde; es kann allerdings auch manuell ein anderer Pfad angegeben werden.

Der in Mutt integrierte Pager unterstützt von sich aus keine HTML-Emails. Man kann sich jedoch leicht behelfen, indem man `lynx` als Pager für HTML-Emails nutzt. Bei den obigen Einstellungen kann die aktuelle Email vom Pager aus mit `lynx` durch Drücken von `l` betrachtet werden. Dabei kann `PageUp` und `PageDown` für ein seitenweises Durchblättern der Email, oder `Ctrl p` und `Ctrl n` für ein zeilenweises Scrollen verwendet werden. Mit `Q` oder `q` wird `lynx` wieder beendet.[2] Als Alternative dazu kann eine HTML-Email auch wie ein Anhang gespeichert und mit Firefox oder einem anderen Webbrowser geöffnet werden.

Compose-Modus

Hat man mit dem Editor eine Email verfasst und den Editor wieder beendet, so gelangt man in das so genannte „Compose“-Fenster. Hier können folgende Funktionen durch Drücken der jeweiligen Taste aufgerufen werden:

[2] Mag man Text aus einer HTML-Email in der Antwort-Email zitieren, so muss dieser von Hand in die Zwischenablage kopiert und in die Antwort-Email eingefügt werden. Meist werden Emails allerdings in reiner Textform oder in gemischter Text- und HTML-Form verschickt; bei diesen Emails funktioniert das automatische Zitieren der Original-Email in der Antwort problemlos.

`s`	Text in Betreffszeile ändern („subject“)
`Esc f`	Text im Absender-Feld ändern („from“)
`c`	Weitere für alle sichtbare Empfänger hinzufügen („copy“)
`b`	Versteckte Empfänger hinzufügen („blind copy“)
`p`	PGP-Verschlüsselungs-Einstellungen vornehmen
`a`	Anhänge an die Email hinzufügen („append“)
↓ oder `j`	Zum nächsten Anhang gehen
↑ oder `k`	Zum vorherigen Anhang gehen
`Enter`	Emailtext beziehungsweise Anhang im Pager betrachten
`e`	Emailtext beziehungsweise Anhang mit Editor öffnen („edit“)
`D`	Als Anhang vorgesehene Datei wieder löschen („delete“)
`y`	Email versenden

Durch Drücken der `Tab`-Taste werden eingegebene Email-Adressen jeweils anhand des Adressbuchs vervollständigt, mittels `Ctrl g` kann die Eingabe abgebrochen werden.

Datensicherung

Sichere Datenübertragung mit `ssh` und `gpg`

Im folgenden Abschnitt sollen das Datenübertragungs-Protokoll SSH sowie die Dateiverschlüsselung GPG anhand von einigen praktischen Beispielen vorgestellt werden.

SSH – Arbeiten auf entfernten Rechnern

SSH steht für „Secure Shell“ und ermöglicht sichere, verschlüsselte Netzwerkverbindungen mit anderen Computern.

Um sich – beispielsweise in einem Heim-Netzwerk – wechselseitig auf einem entfernten Rechner einloggen zu können, sollten auf beiden Rechnern folgende Pakete installiert werden:

```
sudo aptitude install openssh-client openssh-server
```

Hat man hingegen mehrere Clients, die unter einem jeweiligen Benutzernamen Zugriff auf einen Server haben sollen, so genügt es, bei diesen `openssh-client` und auf dem Server `openssh-server` zu installieren. Durch die Installation wird automatisch der SSH-Hintergrund-Dienst `sshd` gestartet; nach jedem Neustart des Rechners ist dieser ebenfalls aktiv.

Um sich von einem Client aus in einem externen Rechner einzuloggen, kann man in einer Shell folgendes eingeben:

```
ssh benutzername@ip-adresse
```

Die Adresse des eigenen Rechners im lokalen Netzwerk kann man sich mittels `ip r` oder `hostname --all-ip-addresses` anzeigen lassen; sie lautet beispielsweise `192.168.2.103`. Daraus kann man entnehmen, dass der Router lokale Netzwerk-Adressen von `192.168.2.1` bis `192.168.2.255` vergibt. Die Netzwerkadressen von anderen Rechnern im lokalen Netzwerk kann man dann beispielsweise erhalten, indem man in einer Shell beispielsweise `nmap -sP 192.168.2.0/24` oder (als Superuser) `nast -m` eingibt.

Existiert für den angegebenen Benutzername auf dem Rechner und wird das Passwort (des Benutzers auf dem externen Rechner) richtig eingegeben, so werden alle folgenden Anweisungen auf dem entfernten Rechner ausgeführt, nur die Bildschirmausgabe erfolgt lokal. Durch Eingabe von `exit` kann die SSH-Sitzung wieder verlassen werden.

`scp`: Datenaustausch im lokalen Netzwerk

Mittels `scp` kann via SSH eine einzelne Datei (oder ein einzelnes Verzeichnis) auf den entfernten Rechner kopiert werden. Die Syntax dafür lautet:

```
scp lokale-datei benutzername@ip-adresse:zielpfad
```

Um mehrere Dateien auf einmal zu kopieren, können diese entweder mittels *tar* gebündelt oder in ein Verzeichnis kopiert werden; mittels `scp -r` kann dieses Verzeichnis dann (wie eine einzelne Datei) rekursiv kopiert werden. Manche Dateimanager wie *Midnight Commander* nutzen ebenfalls intern `scp`, um Dateien auf einfache Weise vom lokalen auf einen externen Rechner oder umgekehrt zu kopieren; der Benutzer spart sich dabei Tipp-Arbeit.

`ssh -X`: Starten von GUI-Programmen

Möchte man auf dem entfernten Rechner nicht nur Shell-Programme, sondern auch Programme mit graphischer Bedienoberfläche (GUI) aufrufen, kann `ssh` mit der Option `-X` aufrufen werden, also mittels:

```
ssh -X benutzername@ip-adresse
```

In diesem Fall bekommt man nach einer richtigen Passwort-Eingabe ebenfalls ein Shell-Fenster angezeigt; aus diesem heraus lassen sich allerdings auch Anwendungen mit graphischen Bedienoberflächen starten, die dann lokal auf dem Monitor angezeigt werden (so, als würden GUI-Programme des eigenen PCs aus der Shell heraus gestartet).

Damit dies funktioniert, muss allerdings auf dem entfernten Rechner folgender Eintrag in der Konfigurations-Datei `/etc/ssh/sshd_config` vorhanden sein:

```
# File: /etc/sshd/sshd_config

X11Forwarding yes
```

War dieser Eintrag noch nicht vorhanden oder das X11-Forwarding bislang deaktiviert, so muss der `ssh`-Dienst auf dem entfernten Rechner mittels `/etc/init.d/ssh restart` nochmal neu gestartet werden.

Anmeldung mit Public Key anstelle eines Passworts

Sicherer als Passwörter sind für die Authentifizierung eines Benutzers so genannte Schlüsselpaare: Ein privater Schlüssel auf dem eigenen PC, und ein öffentlicher Schlüssel, der an beliebig viele andere Stellen kopiert werden kann. Ein Login ist damit nur noch dann möglich, wenn der private und der öffentliche Schlüssel zusammenpassen.

Ein Schlüsselpaar kann in einer Shell folgendermaßen erzeugt werden:

```
mkdir ~/.ssh

ssh-keygen -t rsa
```

Zu Übungszwecken kann bei der Nachfrage nach einem Passwort einfach `Enter` eingegeben werden; der private Schlüssel wird somit nicht mit einem Passwort versehen.

Durch den Aufruf von `ssh-keygen` werden im Verzeichnis `~/.ssh` zwei Dateien angelegt: Die Datei `id_rsa` enthält den privaten Schlüssel, der nicht in falsche Hände geraten sollte, und die Datei `id_rsa.pub` („public“) den öffentlichen Schlüssel, der auf alle Rechner kopiert werden kann, auf denen man sich via SSH einloggen will.

Zum Kopieren des öffentlichen Schlüssels kann folgendes in einer Schell eingegeben werden:

```
ssh-copy-id benutzername@ip-adresse
```

Hierbei muss nochmals das Passwort des Benutzers auf dem Zielsystem eingegeben werden. Durch den Aufruf von `ssh-copy-id` wird der Standard-Schlüssel (oder durch Angabe von `-i pfad` eine explizit angegebene Schlüsseldatei) auf dem Zielrechner der Datei `~/.ssh/authorized_keys` hinzugefügt.

Gibt man anschließend `ssh benutzername@ip-adresse` ein, so erfolgt das Einloggen via Schlüsselpaar anstelle der Eingabe eines Passworts.[1]

Für ein wechselseitiges Verbinden zweier Rechner mittels SSH muss das oben beschriebene Verfahren auf beiden Rechnern erfolgen.

Aliases für häufige Login-Adressen

In der Datei `~/.ssh/config` können Kurzbezeichnungen für häufig besuchte externe Rechner vergeben werden. Um beispielsweise auf einen „Server“ im Home-Netzwerk mit der lokalen Netzwerkadresse `192.168.1.100` zuzugreifen, fügt man der Datei `~/.ssh/config` folgenden Eintrag hinzu:

```
Host server
    HostName 192.168.1.100
    User benutzername
    IdentityFile ~/.ssh/id_rsa
```

[1] Man kann in einem Heim-Netzwerk sogar, um die Sicherheit zu erhöhen, das Anmelden mittels Passworteingabe komplett verbieten. Dazu müssen (mit Superuser-Rechten) in der Datei `/etc/ssh/sshd_config` folgende Einträge vorgenommen werden:

```
PasswordAuthentication no
UsePAM no
```

Damit können sich nur noch Benutzer einloggen, deren öffentliche Schlüssel in der jeweiligen Datei `~/.ssh/authorized_keys` stehen.
In der gleichen Datei sollte zudem ein Login als Root unbedingt verboten werden:

```
PermitRootLogin no
```

Gegebenenfalls kann ein *Benutzer mit Superuser-Rechten* immer noch mit `sudo` systemweite Änderungen vornehmen oder sich mit `sudo su root` dauerhaft Superuser-Rechte verschaffen.

Anschließend muss man nicht mehr `ssh benutzername@192.168.1.100` eingeben, um sich mit dem Server zu verbinden: Von nun an genügt es `ssh server` einzugeben.

Passwortschutz für private Schlüssel

Gelangt der private Schlüssel an eine eigentlich unbefugte Person, so kann sich auch diese ebenso unmittelbar wie ungewollt auf dem Zielrechner einloggen. Um zu verhindern, dass der alleinige „Besitz“ des privaten Schlüssels ausreicht, kann man diesen mit einem Passwort versehen; bevor er für das Einloggen verwendet werden kann, muss er erst mittels des Passworts freigegeben werden.

Üblicherweise werden passwortgeschützte SSH-Schlüssel in Verbindung mit `ssh-agent` genutzt. Dieses Programm wird im Allgemeinen automatisch mit dem X-Server und/oder zu Begin einer Login-Shell gestartet und bleibt aktiv, bis sich der Benutzer wieder abmeldet. Beim der erstmaligen Verwendung des Schlüssels in einer laufenden Sitzung muss das Schlüssel-Passwort eingegeben werden; alle weiteren Zugriffe auf den Schlüssel sind anschließend erlaubt.[2]

Ein passwortgeschützter Schlüssel, beispielsweise `~/.ssh/id_rsa` kann folgendermaßen zur Schlüsselverwaltung mittels `ssh-agent` hinzugefügt werden:

```
ssh-add ~/.ssh/id_rsa
```

Wird `ssh-agent` ohne die explizite Angabe eines Schlüsselpfads gestartet, so werden automatisch alle im Verzeichnis `~/.ssh` liegenden Schlüssel hinzugefügt. Mittels `ssh add -l` können die von `ssh-agent` verwalteten Schlüssel angezeigt werden.

GPG – Signieren und Verschlüsseln von Dateien

GPG steht für „GNU Privacy Guard“ und ist die wohl am weitesten verbreitete Implementierung von PGP („Pretty Good Privacy“). Letzteres stellt einen Standard dar, mit dem u.a. Emails verschlüsselt verschickt werden können, sofern die Software sowohl beim Sender wie auch beim Empfänger installiert ist.

GPG gehört bei fast allen Linux-Distributionen zum Standard, muss also nicht extra installiert werden.

GPG nutzt – ebenso wie SSH – zum Verschlüsseln der Daten ein Schlüsselpaar: Der private Schlüssel bleibt auf dem eigenen Rechner und kann zum Entschlüsseln von Nachrichten verwendet werden; der öffentliche Schlüssel hingegen wird üblicherweise frei verteilt. Mit ihm können Nachrichten an den Eigentümer des Schlüssels verschlüsselt, jedoch nicht entschlüsselt werden.

Erstellen eines GPG-Schlüsselpaars

Zum Erstellen eines neuen Schlüsselpaars gibt man in einer Shell folgendes ein:

[2] Die Freigabe gilt auch für andere Programme, sofern diese in der laufenden Sitzung vom gleichen Benutzer gestartet wurden.

```
gpg --gen-key
```

Hierbei wird man zunächst nach dem gewünschten Verschlüsselungsverfahren gefragt, wobei die Vorgabe „RSA und RSA“ mit `1` ausgewählt werden kann. Als Schlüssellänge sollte man einen möglichst großen Wert nehmen – 2048 Bit sind ok, 4096 Bit sind sicherer und somit besser. Als letztes muss man angeben, wie lange der Schlüssel gültig bleiben soll. Hier sollte durchaus eine Zeitvorgabe, beispielsweise `1y` für „1 Jahr“ eingegeben werden, da Schlüssel ohne Verfallsdatum auch dann im Umlauf bleiben, wenn beispielsweise die zugehörige Emailadresse nicht mehr existiert oder die Schlüssellänge durch immer schnellere Rechner zu klein geworden ist.[3] Anschließend muss man als eindeutige Benutzerkennung noch den Namen und die Emailadresse angeben, zu dem der Schlüssel gehören soll.

Das Passwort beziehungsweise, das man für den Schlüssel vergibt, sollte zwar gut merkbar, aber zugleich ausreichend sicher sein:

- Das Passwort sollte nicht leicht zu erraten sein
- Das Passwort darf Sonderzeichen beinhalten (muss es aber nicht)
- Klein- und Großbuchstaben innerhalb des Passworts werden als unterschiedliche Buchstaben angesehen.
- Lange Passwörter sind in der Regel sicherer als kurze, aber kryptische und damit schwer zu merkende Passwörter (siehe xkcd).
- Als Passwort kann auch ein ganzer Passwort-Satz (ein „Mantra“) verwendet werden, da auch Leerzeichen im Passwort erlaubt sind.

Ohne einen Passwortschutz des privaten Schlüssels könnte jede Person, die Zugriff auf die Schlüsseldatei bekommt, alle mit dem zugehörigen öffentlichen Schlüssel gesicherten Dateien öffnen. Zudem sollte der private Schlüssel nicht verloren gehen: Eine Sicherheits-Kopie des privaten Schlüssels auf eine externe *verschlüsselte Partition* (beispielsweise eine USB-Stick oder eine externe Festplatte) ist dringend zu empfehlen!

Erstellen eines Widerrufungs-Zertifikats

Nachdem man ein Schlüsselpaar erzeugt hat, sollte man gleich im Anschluss daran eine Widerrufsurkunde erzeugen. Mit einer Widerrufsurkunde kann man seinen Schlüssel bei Bedarf als ungültig markieren. Ein solcher Fall kann beispielsweise eintreten, wenn man das Passwort vergessen hat oder der private Schlüssel aufgrund eines Festplatten-Defekts nicht mehr zugänglich ist.

Mit einem als ungültig markierten Schlüssel können keine weiteren Emails mehr verschlüsselt oder signiert werden; bestehende Nachrichten hingegen können weiterhin entschlüsselt werden.

Das Widerrufungs-Zertifikat ist vor allem von Bedeutung, wenn der öffentliche Schlüssel auf einen *Schlüssel-Server* hochgeladen wurde: Diese synchronisieren in regelmäßigen Abständen ihre Datenbanken. Wird ein Schlüssel auf *einem* Server hinzugefügt, so ist er bald

[3] Die Gültigkeitsdauer eines Schlüssels kann jederzeit verlängert werden (siehe Abschnitt *Schlüssel-Attribute ändern*).

auch auf allen anderen Servern zu finden. Würde man seinen Schlüssel nur von einem der Schlüssel-Server löschen, so würde dieser sich nach der nächsten Synchronisation trotzdem wieder auf dem Server befinden. Anstelle einen Schlüssel zu löschen, setzt man daher ein Widerrufszertifikat ein.

Um ein Widerrufungs-Zertifikat zu einem GPG-Schlüssel zu erzeugen, gibt man in einer Shell folgendes ein:

```
gpg --gen-revoke vorname.nachname@email.de
```

Die Frage, ob man dieses Zertifikat wirklich erzeugen will, kann man mit `Ja` beantworten; als Grund für den Widerruf kann man beim voreingestellten Wert `1` („Der Schlüssel ist nicht mehr sicher“) bleiben. Optional kann noch ein Grund für den Widerruf eingegeben werden. Sobald man die Eingabe mit dem Passwort des GPG-Schlüssels bestätigt hat, kann man die auf dem Bildschirm erscheinende Ausgabe in eine Textdatei, beispielsweise `Widerrufungszertifikat-SchluesselID.txt` kopieren. Diese sollte – ebenso wie den privaten GPG-Schlüssel – auf einem externen Speichermedium sicher verwahren werden.

Bei Bedarf: Widerrufen eines Schlüssels

Möchte man einen eigenen Schlüssel tatsächlich widerrufen, so muss man ein zum Schlüssel gehörende Widerrufszertifikat importieren. Ein solcher Schritt muss allerdings gut überlegt sein, denn er kann nicht rückgängig gemacht werden.

```
gpg --import /pfad/Widerrufungszertifikat-SchluesselID.txt
```

Um das Widerrufen eines Schlüssels publik zu machen, sollte man den mit dem Widerrufungszertifikat versehenen Schlüssel erneut an einen Schlüsselserver senden.

```
gpg --send-keys Schlüssel-ID
```

Ein so widerrufener Schlüssel kann nicht mehr genutzt werden, um Nachrichten oder andere Dateien zu verschlüsseln oder zu signieren.

GPG-Schlüsselbund verwalten

Um mit GPG-Schlüsseln arbeiten zu können, wird bei der Erstellung eines Schlüsselpaares automatisch auch ein Schlüsselbund für private und ein Schlüsselbund für öffentliche Schlüssel erzeugt; diese werden im Verzeichnis `~/.gnupg` abgelegt.

Schlüssel auflisten

Mit folgender Anweisung kann man alle öffentlichen Schlüssel auflisten, die im eigenen Schlüsselbund („Public Key Ring“) gespeichert sind:

```
# Öffentliche Schlüssel auflisten:
gpg --list-keys
```

Arbeitet man das erste mal mit GPG, so enthält der Public-Key-Ring nur den eigenen öffentlichen Schlüssel. Im Laufe der Zeit kommen dann weitere Schlüssel von anderen Personen oder Einrichtungen hinzu, die ebenfalls GPG verwenden (beispielsweise wenn man einer solchen Person eine mit GPG verschlüsselte Email schreiben möchte).

Alle öffentlichen Schlüssel sind in der Datei `~/.gnupg/pubring.gpg` abgelegt. private Schlüssel („Secret Keys“) werden hingegen in der Datei `~/.gnupg/secring.gpg` gespeichert. Sie können folgendermaßen angezeigt werden:

```
# Private Schlüssel auflisten:
gpg --list-secret-keys
```

Schlüssel exportieren

Möchte man, dass einem andere Personen eine verschlüsselte Email schreiben können, so müssen diese einen Zugang zum öffentlichen Schlüssel haben. Hierzu kann man den eigenen öffentlichen Schlüssel beispielsweise als Text-Datei auf einer Homepage veröffentlichen oder der jeweiligen Person per Email schicken.

Um einen Schlüssel zu exportieren, kann man `gpg` mit der Option `--export` aufrufen. Hierbei werden allerdings standardmäßig *alle* Schlüssel ausgegeben. Möchte man nur den eigenen Schlüssel exportieren, so muss man die Auswahl dadurch einschränken, indem man hinter `--export` beispielsweise die zum Schlüssel gehörende Email-Adresse, den zum Schlüssel gehörenden Namen oder die ersten Stellen der Schlüssel-ID angibt:

```
# Eigenen Schlüssel exportieren:
gpg -a -o ~/my-public-key.txt --export vorname.nachname@email.de
```

Über die Option `-o` wird der Name der Datei festgelegt, in die der Schlüssel exportiert werden soll. Die Option `-a` als Kurzform für `--armor` bewirkt, dass der Schlüssel ausschließlich durch ASCII-Symbole dargestellt wird, so dass die resultierende Datei ohne Probleme auf eine Homepage hochgeladen oder als Anhang per Email verschickt werden kann (manchmal wird auch die Datei-Endung `.asc` oder `.gpg` anstelle von `.txt` verwendet).

Schlüssel importieren

Mit folgender Anweisung kann ein öffentlicher Schlüssel aus einer Textdatei importiert werden:

```
# Schlüssel aus Datei importieren:
gpg --import schluesseldatei
```

Die Schlüsseldatei muss hierbei mitsamt Endung (meist `.txt`, `.asc` oder `.gpg`) angegeben werden.

Sobald ein Schlüssel zum eigenen Schlüsselbund hinzugefügt wurde, kann er verwendet werden, um beispielsweise der zugehörigen Person verschlüsselte Daten zu senden.

Mittels `gpg --delete-key` kann man den öffentlichen Schlüssel mit der angegebenen ID wieder aus dem eigenen Schlüsselbund entfernen:

```
# Schlüssel aus Schlüsselbund entfernen:
gpg --delete-key SchluesselID
```

Mittels `gpg --delete-secret-key ID` kann man entsprechend einen privaten Schlüssel löschen.

Schlüssel-Attribute ändern

Mittels folgender Anweisung lassen sich auch nachträglich Änderungen an einem GPG-Schlüssel vornehmen:

```
# Schlüssel-Attribute editieren:
gpg --edit-key SchluesselID
```

Nach Eingabe dieser Anweisung erscheint ein `gpg>`-Eingabe-Prompt, in dem weitere GPG-Anweisungen aufgerufen werden können

- Mit `help` bekommt man eine Liste aller möglichen Anweisungen mitsamt kurzen Beschreibungen angezeigt.
- Mit `expire` kann man das Verfalls-Datum (`expire`) eines eigenen Schlüssels ändern
- Mit `passwd` kann man das Passwort (Mantra) eines eigenen Schlüssels ändern

Mit `quit` kann das Editieren wieder beendet werden.

Nutzung von Schlüssel-Servern

Eine weit verbreitete Methode, öffentliche GPG-Schlüssel publik zu machen, ist die Verwendung von so genannten Schlüssel-Servern. Man kann beispielsweise den (beziehungsweise einen) eigenen Schlüssel auf einen solchen Server hochladen, so dass er von anderen GPG-Nutzern in der ganzen Welt gefunden werden kann. Umgekehrt kann man so auch nach den GPG-Schlüsseln von anderen Personen suchen.

Um einen Standard-Schlüssel-Server festzulegen, kann man in der Konfigurationsdatei `~/.gnupg/gpg.conf` beispielsweise folgenden Eintrag vornehmen:

```
# Eintrag in der Datei ~/.gnupg.conf :
keyserver sks-keyservers.net
```

Für die Nutzung von Schlüssel-Servern gibt es dann folgende `gpg`-Anweisungen:

- Mit `gpg --search Name` oder `gpg --search emailadresse` kann man nach öffentlichen Schlüsseln anderer Personen suchen. Sind Einträge vorhanden, so werden diese automatisch nummeriert aufgelistet. Man erhält daraufhin eine Abfrage, ob man einen der gefundenen Schlüssel durch Eingabe der zugehörigen Nummer in den eigenen Schlüsselbund importieren möchte, oder ob man die Eingabe mittels `b` beenden möchte.

- Mit `gpg --refresh` werden alle öffentlichen Schlüssel aktualisiert. Auf diese Weise erhält man beispielsweise Informationen darüber, ob ein Schlüssel widerrufen wurde, andere Schlüssel-Signaturen hinzugekommen sind, usw.
- Mit `gpg --recv SchluesselID` wird der öffentliche Schlüssel mit der angegebenen ID direkt importiert. Ist der angegebene Schlüssel bereits im Schlüsselbund enthalten, so wird der Schlüssel aktualisiert.
- Mit `gpg --send SchluesselID` wird der öffentliche Schlüssel mit der angegebenen ID an den Schlüssel-Server geschickt; `gpg` gibt hierbei eine Fehler- oder Erfolgsmeldung auf dem Bildschirm aus.

  Achtung: Schlüssel, die einmal auf einen Schlüssel-Server hochgeladen wurden, können nie wieder gelöscht werden!

Wie bereits im Abschnitt *Widerrufungs-Zertifikat* erwähnt, gibt es keine Möglichkeit, einen Schlüssel wieder von einem Schlüssel-Server zu löschen. Schlüssel verlieren nur ihre Gültigkeit, wenn sie verfallen oder widerrufen werden; das Erstellen eines Widerrufungs-Zertifikats ist daher dringend zu empfehlen, bevor ein öffentlicher Schlüssel an andere weitergegeben wird – schließlich kann *jeder* öffentliche Schlüssel auf einen Schlüssel-Server hochladen!

Daten signieren

Signaturen sollen als „digitale Unterschriften“ die Authentizität beispielsweise einer Nachricht beweisen. Würden Nachrichten von offizieller Seite konsequent signiert, wäre es deutlich schwerer, mit gefälschten Nachrichten Unruhe oder Schaden zu verursachen.

Ein Signatur einer Nachricht wird unter Verwendung des privaten Schlüssels erzeugt; diese kann dann vom Empfänger anhand des öffentlichen Schlüssels des Senders überprüft werden. Dabei wird nicht nur der Absender überprüft (nur dieser hat den privaten Schlüssel), sondern auch, ob der Text unverändert angekommen ist.

Eine Datei kann mit einer der folgenden Anweisungen signiert werden:

- Mit `gpg -s` beziehungsweise `gpg --sign` kann man eine Datei mit seinem privaten Schlüssel signieren; sie wird dabei gleichzeitig komprimiert und ist somit nicht mehr ohne weiteres lesbar.
- Mit `gpg --clearsign` kann man eine Datei signieren, wobei die Datei nicht komprimiert wird und somit lesbar bleibt.
- Mit `gpg -b` beziehungsweise `gpg --detach-sign` wird die Signatur in einer separaten Datei abgelegt. Diese Variante ist beispielsweise zum Signieren von Archiv-Dateien empfehlenswert (diese können allerdings auch ohne Überprüfung der Signatur entpackt werden).

Bei allen obigen Anweisungen kann die Option `-a` beziehungsweise `--armor` nützlich sein, um reine ASCII-Zeichen zu erzwingen; dies ist insbesondere beim Versenden von signierten und/oder verschlüsselten Daten via Email empfehlenswert.

Die Signatur einer (unverschlüsselten) Datei kann man folgendermaßen prüfen:

```
# Signatur überprüfen:
gpg --verify Datei
```

Wurde die Datei von einer anderen Person signiert, so muss man zur Überprüfung der Signatur den öffentlichen Schlüssel dieser Person im eigenen Schlüsselbund haben.

Das „Web of Trust“

Eine prinzipielle Schwachstelle der Public-Key-Methode ist die Verbreitung der öffentlichen Schlüssel: Ein Angreifer könnte einen öffentlichen Schlüssel mit einer falschen User-ID in Umlauf bringen. Wenn ein Sender diesen Schlüssel zur Verschlüsselung einer Nachricht verwendet und der Angreifer die Nachricht abfangen kann, so kann er diese dekodieren und lesen. Wenn der Angreifer die Nachricht anschließend mit einem echten öffentlichen Schlüssel kodiert an den eigentlichen Empfänger weiterleitet, fällt der Angriff unter Umständen nicht einmal auf. Derartige potentielle Angriffe, die auch bei anderen Verschlüsselungsverfahren möglich sind, nennt man daher „Man-in-the-Middle“-Angriffe.

Eine von von PGP beziehungsweise GnuPG gewählte Strategie gegen derartige Angriffe besteht im *Signieren der Schlüssel*: Jeder öffentliche Schlüssel kann von anderen Personen unterschrieben werden. Eine solche Unterschrift soll wiederum bestätigen, dass der Schlüssel authentisch ist, also zu der in der User-ID angegebenen Person gehört.

Jeder Benutzer muss selbst entscheiden, welchen Unterschriften er wie weit traut. Man kann einen Schlüssel beispielsweise dann als vertrauenswürdig einstufen, wenn er von einer Person unterzeichnet wurde, der man vertraut. Ebenso sollte man selbst einen anderen Schlüssel nur dann unterzeichnen, wenn man die zum Schlüssel gehörende Person kennt und sich der Authentizität des Schlüssels auch wirklich sicher ist.

Daten verschlüsseln und entschlüsseln

Beim Verschlüsseln von Daten gibt es nun zwei Möglichkeiten:

- Möchte man eine Datei mit dem eigenen privaten Schlüssel verschlüsseln (so dass man sie also nur selbst wieder entschlüsseln kann), so wird ohne weitere Vorgaben der Standard-Schlüssel verwendet; ebenso kann man mit der Option `-u UserID` gezielt einen Schlüssel auswählen.[4]
- Möchte man eine Datei verschlüsselt an einen anderen Empfänger („Recipient“) schicken, so kann der Schlüssel dieses Empfängers mittels der Option `-r UserID` ausgewählt werden.

Die eigentliche Anweisung zum Verschlüsseln einer Datei lautet:

[4] Hat man mehrere private Schlüssel, so kann man in der Konfigurationsdatei `~/.gnupg/gpg.conf` auch folgendermaßen einen Standard-Schlüssel festlegen:

```
# Eintrag in der Datei ~/.gnupg.conf :
default-key SchluesselID
```

```
# Datei verschlüsseln:
gpg -e Empfaenger Datei
```

Hierbei steht `-e` als Kurzform für die gleichbedeutende Option `--encrypt`.

Zusätzlich zu einer Verschlüsselung ist es stets sinnvoll, die Datei auch noch zu signieren.

```
# Datei verschlüsseln und signieren:
gpg -u SenderID -r EmpfaengerID --armor --sign --encrypt Datei

# Datei verschlüsseln und signieren (Kurzform):
gpg -u SenderID -r EmpfaengerID -a -s -e Datei
```

Das rekursive Verschlüsseln eines Verzeichnisses mitsamt Unterverzeichnissen ist nicht möglich; hierfür muss zunächst beispielsweise mittels *tar* oder *zip* eine Archiv-Datei des Verzeichnisses erstellt werden.

Wenn eine verschlüsselte Datei signiert ist, so wird beim Entschlüsseln die Signatur automatisch mit geprüft. Die Anweisung zum Entschlüsseln einer Datei lautet:

```
# Datei entschlüsseln:
gpg -d Datei
```

Hierbei steht `-d` als Kurzform für die gleichbedeutende Option `--decrypt`.

Ohne weitere Vorgaben wird der Inhalt der Datei auf dem Standard-Ausgabe-Kanal `stdout` ausgegeben, üblicherweise also auf dem Bildschirm; mittels der Option `-o Ausgabedatei` wird die Ausgaben stattdessen in die angegebene Datei geschrieben.

Grenzen der Sicherheit

Die von GPG verwendeten Algorithmen gelten allgemein bis heute als nicht knackbar; dies bedeutet jedoch nicht automatisch, dass alle mit GPG verschlüsselten Daten tatsächlich sicher sind. Bekommt beispielsweise ein professionaler Angreifer (beispielsweise über einen Trojaner) Zugriff auf einen privaten Schlüsselbund, der nur mit einem schlechten Passwort gesichert ist, so kann er dieses weitaus leichter knacken als die mit GPG verschlüsselten Daten selbst; mit dem privaten Schlüssel lassen sich die Daten allerdings regulär entschlüsseln. Für die Datensicherheit ist somit auch die allgemeine Systemsicherheit von Bedeutung.

Wie „sicher“ ein Verschlüsselungs-System sein muss, hängt letztlich vom konkreten Anwendungsfall ab: Möchte man „unkritische“ private Daten nur vor einem einfachen Fremdzugriff schützen (ähnlich wie das bei Verwendung eines Kuverts bei einem Brief der Fall ist), so wird man sich wohl weniger Gedanken über Sicherheit machen als wenn man beispielsweise Server-Passwörter, Bank-Daten oder ähnlich sensible Daten verschlüsseln beziehungsweise vor Angreifern schützen möchte.

Auch ohne Verschlüsselung ist der Einsatz von GPG sinnvoll, um beispielsweise bei Quellcode-Paketen die Integrität der jeweiligen Daten sicherzustellen (also ausschließen zu

können, dass die jeweiligen Pakete nicht durch Dritte manipuliert wurden); nicht umsonst gehört `gpg` daher zum Standard-Umfang jeder Linux-Distribution.[5]

Links

- GPG Wikibook
- GPG-Tutorial
- GPG-Anleitung
- GPG-Handbuch
- Seahorse – ein GPG Frontend (Manual, en.)

Datenträger-Verschlüsselung mit `cryptsetup`

Mit Cryptsetup kann man ganze Partitionen mit einem Passwort-Schutz versehen. Dies wird üblicherweise genutzt, um private Daten vor einem Fremdzugriff zu schützen; man kann allerdings, wie im zweiten Teil dieses Abschnitts beschrieben, auch die Partition des Betriebsystems verschlüsseln.

Verschlüsselung von Daten-Partitionen

Die Verschlüsselung einer gewöhnlichen Partition auf einer Festplatte oder einem USB-Stick mit kann man in einer Shell nach dem folgendem Schema erreichen:

1. Zunächst muss man herausfinden, welche Device-Stelle einem Datenträger beziehungsweise einer Partition zugeordnet ist:

[5] Ein bekanntes Verfahren hierfür ist beispielsweise die so genannte „MD5-Prüfsumme“: Nach diesem Verfahren wird zu der relevanten Datei zunächst über eine Hash-Funktion wie beispielsweise MD5 oder (besser) SHA-1 ein Hash-Wert berechnet (unter Linux gibt es dafür die Shell-Programme `md5sum` beziehungsweise `sha1sum`). Dieser Hash-Wert wird anschließend vom Herausgeber mittels seines privaten Schlüssels signiert.

Etwas vereinfacht kann man sich das so vorstellen, als würde man zu einer zehnstelligen Zahl die Quersumme ermitteln, also die Summe aller in der Zahl vorkommenden Ziffern zu berechnen. Das ist nicht wirklich schwer – weitaus schwerer ist es hingegen, von dieser Quersumme auf die ursprüngliche Zahl zu schließen. Eine Hash-Funktion macht letztlich nichts anderes: Sie liefert zu einer bestimmten Dateneingabe stets einen eindeutigen Wert.

Der Empfänger kann die Signatur wiederum mittels des öffentlichen Schlüssels des Herausgebers überprüfen.

- Eine korrekte Signatur kann als Beweis dafür angesehen werden, dass der Hash-Wert tatsächlich vom Herausgeber stammt.
- Anschließend kann geprüft werden, ob die Hash-Funktion bei Anwendung auf die heruntergeladene Datei einen identischen Hash-Wert erzeugt. Ist dies der Fall, so ist die Wahrscheinlichkeit sehr groß, dass die erhaltene Datei tatsächlich der Original-Datei entspricht.

Inzwischen gilt MD5 nicht mehr als sicher; man sollte daher besser auf einen SHA-Algorithmus zurückgreifen.

```
# Partitionen und Einhängepunkte auflisten:
lsblk
```

Angenommen, in einem Rechner ist eine Festplatte enthalten, auf der sich drei Partitionen befinden: Eine Partition für das Basis-System `/`, eine für das Home-Verzeichnis `/home` und eine als Swap (erweiterter Arbeitsspeicher). Der obige Befehl würde die Partitionen dann als `/dev/sda1`, `/dev/sda2` und `/dev/sda3` anzeigen.

Weitere Festplatten, USB-Sticks usw. erhalten fortlaufend die Device-Bezeichnung `/dev/sdb`, `/dev/sdc`, `/dev/sdd`, usw. Sofern Partitionen auf den Datenspeichern vorhanden sind, gibt eine angehängte Nummer die jeweilige Partitionsnummer auf dem Datenträger an.

2. Als nächstes sollte eine neue Partition mit einem Partitions-Programm erstellt werden.

   ```
   sudo gparted /dev/sdX
   ```

   Hierbei muss anstelle `X` die konkrete Device-Bezeichnung angegeben werden.

   Achtung: Partitionierungen sind immer mit der Gefahr eines Datenverlusts verbunden. Es ist dringend empfehlenswert, vorab eine Sicherheits-Kopie der auf dem Datenträger gespeicherten Daten zu erstellen!

   Ist auf dem Datenträger keine Partitionstabelle vorhanden (dies erkennt man unter anderem daran, dass unter der Rubrik `Partition` keine neue Partition erstellt werden kann), so muss mittels `Gerät -> Partitionstabelle erstellen` erst eine neue Partitionstabelle angelegt werden; dabei sollte als Typ `gpt` gewählt werden.

   Nun kann über `Partition -> Neu` eine neue Partition angelegt werden; als Dateisystem sollte `ext4` gewählt werden.

3. Anschließend kann eine LUKS-Partition mit `cryptsetup` angelegt werden:

   ```
   sudo cryptsetup -c aes-xts-plain -y -s 512 luksFormat /dev/sdX1
   sudo cryptsetup luksOpen /dev/sdX1 crypt_name
   sudo mkfs.ext4 /dev/mapper/crypt_name
   ```

   Hierbei muss wiederum anstelle von `X1` die konkrete Device-Bezeichnung angegeben werden. Innerhalb des LUKS-Containers befindet sich anschließend eine gewöhnliche `ext4`-Partition.

4. Die neue Partition kann nun in einen beliebigen Verzeichnispfad eingebunden werden:

   ```
   sudo mkdir /media/crypt
   sudo mount /dev/mapper/crypt_name /media/crypt
   ```

Die Partition kann von diesem Moment an über den entsprechenden Verzeichnispfad (im Beispiel `/media/crypt`) wie ein großer Daten-Ordner genutzt werden.

Einbinden bestehender LUKS-Partitionen

Moderne Linux-Systeme wie Linux Mint erkennen automatisch verschlüsselte Datenträger (z.B. USB-Sticks oder externe Festplatten) und öffnen ein entsprechendes Dialog-Fenster zur Passworteingabe.

Von Hand lässt sich eine bestehende LUKS-Partition nach folgendem Schema in ein laufendes System einbinden („mounten“):

```
sudo cryptsetup luksOpen /dev/sdX1 crypt_name
sudo mount /dev/mapper/crypt_name /media/crypt
```

Hierbei muss wiederum anstelle von `X1` die konkrete Device-Bezeichnung angegeben werden. Sowohl der beim Öffnen der Partition vergebene Crypt-Name als auch der beim Mounten festgelegte Einhänge-Punkt sind frei wählbar.

Das Aushängen einer – von keinem Programm benutzten – LUKS-Partition erfolgt durch ein Anklicken des Datenträger-Icons auf dem Desktop mit der rechten Maustaste oder in einem Shell-Fenster nach folgendem Schema:

```
sudo umount /media/crypt
sudo cryptsetup luksClose /dev/mapper/crypt_name
```

Ein Herunterfahren des Systems bewirkt ebenfalls ein Aushängen und Verschließen aller eingehängten Partitionen.

Wird eine Partition ausschließlich von einem Nutzer verwendet, so empfiehlt sich als Mount-Pfad anstelle `/media/crypt` besser ein Ordner im Unterverzeichnis des Benutzers, beispielsweise `/media/tux/crypt`. Der Benutzer hat dann auch ohne Root-Rechte vollen Lese- und Schreibzugriff auf alle Daten der Partition.

Verschlüsselung der System-Partition

Mittels Cryptsetup und Luks können nicht nur „normale“ Partitionen verschlüsselt werden; es ist während einer Linux-Installation auch möglich das gesamte System bis auf einen notwendigen Boot-Bereich zu verschlüsseln.[1] Grundlegende Linux- beziehungsweise *Shell*-Kenntnisse sollten hierzu allerdings vorhanden sein.

[1] Eine Verschlüsselung des Betriebsystems ist nur dann sinnvoll, wenn ein Angriff mit physischem Zugang zum Rechner zu befürchten ist. Ist ein mit dieser Methode verschlüsselter Rechner ausgeschaltet, so ist er wohl bestmöglich geschützt. Erlangt ein Angreifer allerdings im laufenden Betrieb Administrator-Rechte, beispielsweise durch mögliche Sicherheitslücken bei Server-Anwendungen, so hilft auch die (im laufenden Zustand bereits geöffnete) System-Verschlüsselung nicht weiter. Die Methode zeigt allerdings einmal mehr, was für „Tricks“ auf Linux-Systemen grundsätzlich möglich sind..

Persönlich nutze ich eher die bereits beschriebene Methode der *Partitions-Verschlüsselung* und sichere private Daten zudem auf Offline-Speichermedien. Das Programm *keepassx* ist zur geschützten Verwaltung von guten, also ausreichend langen Passwörtern ebenfalls sehr zu empfehlen.

Nötige Partitionen erstellen und verschlüsseln

Vor der Installation werden zwei primäre Partitionen angelegt. Sie lassen sich beispielsweise bei einer Linux Mint-Installation mittels des graphischen, leicht bedienbaren und bereits auf der Live-CD enthaltenen Programms `gparted` erstellen:

Partition	Name	Größe
eine Boot-Partition	`/dev/sda1`	300 bis 500 MB
eine restliche Partition	`/dev/sda2`	min. 15 GB

Die obigen Partitionsnamen können auch vertauscht sein, entscheidend ist zu wissen, welche jeweils gemeint ist. Als Formatierung verwende ich am liebsten das schnelle, sichere und wartungsarme Dateisystem `ReiserFS`. In Anlehnung an die erprobte Original-Anleitung (en.) werden nun in einem Terminal als Superuser (`su` eingeben!) nacheinander folgende Schritte durchlaufen:

1. Formatierung der Boot-Partition:

```
mkfs.reiserfs -l boot /dev/sda1
```

2. Anlegen eines verschlüsselten System-Devices:

```
cryptsetup luksFormat --cipher aes-cbc-essiv:sha256 /dev/sda2
cryptsetup luksOpen /dev/sda2 sda2_crypt
```

3. Aufteilung des Crypt-Devices in zwei Bereiche (`logical volume`): Einen `swap`-Bereich, welcher dem System zur Auslagerung von Dateien dient (erweiterter Arbeitsspeicher, 1 bis 4 GB), sowie die eigentliche Systempartition `root` mit dem restlichen Festplattenspeicher:

```
pvcreate /dev/mapper/sda2_crypt
vgcreate cryptVG /dev/mapper/sda2_crypt
lvcreate -n swap -L 4G cryptVG
lvcreate -n root -l 100%FREE cryptVG
```

4. Formatierung der neuen Bereiche:

```
mkswap -L swap /dev/cryptVG/swap
mkfs.reiserfs -l root /dev/cryptVG/root
```

Installations-Routine und nachträgliche Anpassungen

Nun kann der Installations-Assistent gestartet werden. Hierzu klickt man das entsprechende Icon auf dem Desktop an und füllt die nötigen Felder (gewünschter Benutzername, Passwörter, Zeitzone, Tastaturlayout, etc.) aus. Im Partitions-Auswahlmenü ist darauf zu achten, dass die Bereiche richtig eingebunden werden:

Partition	Einhängepunkt
/dev/mapper/cryptVG-root	/
/dev/sda1	/boot

Nach dieser Basis-Installation, die Abhängig von der Hardware-Geschwindigkeit zwischen 15 und 30 Minuten dauert, müssen noch folgende Anpassungen vorgenommen werden:

1. Einbinden des neuen Systems:

```
mkdir /media/sidux
mount /dev/cryptVG/root /media/sidux
mount /dev/sda1 /media/sidux/boot
```

2. Drei Dateien müssen nun mit einem Texteditor erstellt beziehungsweise angepasst werden:

   - Die Datei `/media/sidux/etc/crypttab` muss folgendes Schema aufweisen:

     ```
     # target    source_device                          key_file  options
     sda2_crypt  /dev/disk/by-uuid/[UUID of /dev/sda2]  none      luks
     ```

     Die UUID einer Partition bekommt man in einem separaten Terminal-Fenster im Verzeichnis `/dev/disk/by-uuid` mittels `ls -l` (long list) angezeigt. Sie sieht ungefähr so aus: `550e8400-e29b-11d4-a716-446655440000`. Es genügt, die passende UUID mit der Maus zu markieren, um sie im anderen Fenster per mittlerem Mausklick an gewünschter Stelle einfügen zu können.

   - In der `/media/sidux/etc/initramfs-tools/conf.d/cryptroot` (die Datei existiert noch nicht!) muss folgendes eingetragen werden:

     ```
     target=sda2_crypt,source=UUID=[UUID of your /dev/sda2],lvm=cryptVG-
     ↪root
     ```

   - Die Datei `/etc/initramfs-tools/modules` muss noch um folgende Einträge (einen je Zeile) ergänzt werden:

     ```
     aes-i586
     aes-x86_64
     xts
     gf128
     sha256
     ```

3. Nun kann man die Installation durch folgende Kommandos abschließen:

```
chroot /media/sidux
mount -t proc proc /proc
mount -t sysfs sysfs /sys
update-initramfs -u
umount proc
umount sys
```

```
exit
reboot
```

Nach einem Reboot wird nun beim Start ein Passwort verlangt, bevor das System wie gewohnt hochfährt.

Login von Live-Disk

Sollte beim Starten des PCs die verschlüsselte Partition nicht erkannt werden (und damit ein Booten unmöglich sein), so kann das System dadurch zum Laufen gebracht werden, indem man mittels einer Live-Disk (oder einem Live-Stick) bootet und als Superuser folgende Zeilen in einer Shell eingibt:

```
cryptsetup luksOpen /dev/sda2 root
mkdir /media/root
pvscan
lvscan
vgscan
vgchange -ay
```

Damit werden die vorhandenen Partitionen erkannt und aktiviert. Anschließend können sie gemountet werden:

```
mount /dev/cryptVG/root /media/root
mount /dev/sda1 /media/root/boot
mount --bind /dev /media/root/dev
mount --bind /proc /media/root/proc
mount --bind /sys /media/root/sys
```

Nun kann man den Root-Pfad des laufendes Systems auf die gemountete Festplatten-Partition umstellen:

```
chroot /media/root                  # ins Filesystem der Festplatte wechseln..
```

Liegt der Fehler an einer fehlerhaften Einstellung des Bootloaders `grub`, so kann das Problem mit folgender Routine automatisch behoben werden:

```
grub-install --recheck /dev/sda     # neues Einrichten des GRUB
update-grub                         # Partitionen werden erkannt
grub-mkconfig > /boot/grub/menu.lst # Bootmenü wird neu geschrieben
```

Nach einem Reboot sollte der PC wie gewohnt hochfahren. Das Verfahren, mittels der obigen `chroot`-Routine von einem Live-System aus auf das installierte System zu wechseln, kann übrigens auch auf nicht verschlüsselte Systeme angewendet werden.

Daten-Synchronisierung mit `unison` und `bsync`

Unison

Möchte man eine Synchronisierung zweier Verzeichnisse nicht nur in eine Richtung, so ist das Tool Unison beziehungsweise das gleiche Programm mit graphischer Oberfläche `unison-gtk` zu empfehlen. Es lässt sich über die gleichnamigen Pakete via `apt` installieren:

```
sudo apt-get install unison unison-gtk
```

Nach der Installation kann Unison mittels `unison-gtk` aufgerufen werden. Intern verwendet Unison das Shell-Programm *rsync* zur Synchronisierung von Daten. Um Unison zu benutzen, erstellt man ein „Profil“, in welchem man zwei zu synchronisierende Verzeichnisse auswählt. Öffnet man dieses im Auswahlmenü, so scannt Unison die Verzeichnisse automatisch nach Veränderungen und zeigt diese mitsamt der Richtung und der Art der Veränderung graphisch an. Mit einem Klick auf „Go“ (Hotkey `g`) werden die Änderungen übernommen.

`rsync` und `unison` eignen sich sehr gut zur Verwaltung von Sicherheitskopien oder zum „Mitnehmen“ eines Projektes von einem stationären PC auf einen USB-Stick. Veränderungen sind dabei erlaubt, denn sie können wiederum in umgekehrter Richtung synchronisiert werden.

Persönlich verwende ich zur Synchronisierung von Dateien zwischen meinem Rechner und einem (mit LUKS verschlüsselten) USB-Stick folgende Methode: In einem eigenen Verzeichnis namens `shared` lege ich für jede zu synchronisierende Datei oder jeden zu synchronisierenden Ordner einen gleichnamigen Symlink ab, der allerdings am Namen die (zusätzliche) Endung `.sync` erhält. Dieses Verzeichnis mit den entsprechenden Symlinks muss sowohl auf der Festplatte wie auch auf dem USB-Stick vorhanden sein, die Ziele der Symlinks sind allerdings logischerweise unterschiedlich, beispielsweise:

```
/home/user/shared
    code.sync       # ---> /home/user/data/code
    configs.sync    # ---> /home/user/data/configs
    homepage.sync   # ---> /home/user/data/homepage

/media/user/usb0/shared
    code.sync       # ---> /media/user/usb0/code
    configs.sync    # ---> /media/user/usb0/configs
    homepage.sync   # ---> /media/user/usb0/homepage
```

Falls noch nicht vorhanden, so wird anschließend das Verzeichnis `~/.unison` angelegt. In diesem Verzeichnis lassen sich beliebig viele Synchronisations-Profile als Textdateien mit der Endung `.prf` anlegen. Für die Synchronisation mit dem USB-Stick sieht ein solches Profil bei mir folgendermaßen aus:

```
# Datei: ~/.unison/usb-sync.prf

# Quell- und Zielverzeichnis:
root = /home/user/shared
```

```
root = /media/user/usb0/shared

# Angabe der zu synchronisierenden Dateien:
follow = Name *.sync

# Folgende Dateien dennoch ignorieren:
ignore = Regex .*/.backupdir/*
ignore = Regex .*/.git/*
ignore = Regex .*/.hg/*

# Bei Unterschieden zwischen Dateien nur das Nötigste ausgeben:
diff = diff -y -W 79 --suppress-common-lines
```

Diese Variante setzt voraus, dass der USB-Stick immer an der gleichen Stelle eingebunden wird (im obigen Beispiel `/media/user/usb0`). Anschließend muss nur noch `unison usb-sync` aufgerufen werden, um eine Synchronisation der angegebenen Inhalte zu erreichen.

Im Shell-Modus wird die von Unison vorgeschlagene Synchronisationsrichtung mit `<----` oder `---->` angezeigt. Drückt man `f` („follow“), so wird diese Empfehlung übernommen. Wurden sowohl im Quell- wie auch im Zielverzeichnis Änderungen vorgenommen, so zeigt Unison `<-?->` an. Der Benutzer muss in diesem Fall die Unterschiede zwischen den Dateiversionen gegebenenfalls selbst überprüfen (beispielsweise mittels *vimdiff*) und kann anschließend entweder mittels `>` oder `<` eine Synchronisationsrichtung manuell angeben.

Synchronisierungen mit `rsync` beziehungsweise `unison` lassen sich nicht rückgängig machen. Zu solch einem Zweck oder für Mehrbenutzer-Systeme, wenn es zu konkurrierenden Entwicklungen kommen kann (wenn beispielsweise die gleiche Datei in zwei Verzeichnissen auf unterschiedliche Weise verändert wird), sollte eine Versionskontroll-Programm wie `git` oder `mercurial` genutzt werden.

Eine mit der Programmiersprache Python3 geschriebene Neu-Implementierung von Unison (allerdings bislang ohne graphische Bedienoberfläche) heißt bsync.

... to be continuted ...

Backups

... to be continuted ...

Netzwerk-Basics

Grundbegriffe und Konzepte

Im folgenden Abschnitt sollen wichtige Netzwerk-Begriffe kurz allgemein (unabhängig vom Betriebsystem) erklärt werden; auf Linux-spezifische Anwendungen wird dann im übernächsten Abschnitt eingegangen.

Zugriff auf das Internet

Um mit einem Computer ins Internet zu gelangen, ist ein Modem notwendig; dieses „übersetzt“ lokale Netzwerk-Signale in eine „Sprache“, die der jeweilige Internet-Anbieter („Internet Service Provider“, kurz ISP) versteht.[1]

Die einzelnen Geräte, die heutzutage für einen Internet-Zugang von Bedeutung sind, haben folgende Aufgaben:

- Um in einem Haushalt nicht nur mit einem einzelnen, sondern mit mehreren Geräten gleichzeitig Zugang zu haben, wird das Modem im Allgemeinen mit einem so genannten „Router“ verbunden.

  Mittels Routern ist es möglich, mehrere logisch voneinander getrennte Teil-Netzwerke zu definieren.[2] Beispielsweise kann es der Fall sein, dass es in einem Mehrfamilien-Haus nur eine Internet-Verbindung gibt, die von mehreren Familien genutzt werden soll. Jede Familie kann dann einen eigenen Router nutzen, der jeweils ein lokales Netzwerk bereitstellt. Die Router der einzelnen Familien sind wiederum mit dem Haupt-Router verbunden und haben über diesen Zugang zum Internet, nicht jedoch zu den übrigen Teil-Netzwerken. Router können somit – sofern sie einen eingebauten Switch haben – einen einzigen Internet-Zugang in mehrere logisch voneinander getrennte Teil-Zugänge aufteilen.

- Router werden üblicherweise in Kombination mit einer so genannten Firewall betrieben, die das lokale Netzwerk gegen Hacker-Angriffe absichern soll. Früher waren

[1] In den Anfangszeiten des Internets konnte man, ähnlich wie immer noch bei Fax-Geräten, eine Art von akustischen „Morse-Signalen“ bei dieser Datenübertragung hören; inzwischen findet die Datenübertragung meist über optische Signale statt.

[2] Streng genommen regeln Router nur die Kommunikation zwischen verschiedenen Teil-Netzwerken; für die Aufteilung des Datenstroms kommen hingegen „Switch“-Geräte zum Einsatz. Oftmals werden heutzutage allerdings beide Geräte in ein einziges physisches Gerät verbaut, das dann ebenfalls schlichtweg „Router“ genannt wird.

Firewalls stets eigenständige Geräte; inzwischen sind sie oftmals ebenfalls in Router-Geräte integriert und stellen somit zwar nicht aus physischer, wohl aber aus logischer Sicht eigenständige Geräte dar. Zudem gibt es Software-Firewalls, so dass bei größeren Firmen-Netzwerken die Firewall unter Umständen auch heute noch auf einem eigenen Server läuft.

- Um das Internet-Signal schließlich auf mehrere Rechner aufzuteilen, kommt stets ein so genannter „Switch“ zum Einsatz; auch dieser kann unter Umständen bereits in die Box des Routers integriert sein, es gibt jedoch auch eigenständige Switch-Geräte (mit bis zu 48 Anschluss-Ports).

  An einen Switch kann gegebenenfalls auch ein Wireless Access Point angeschlossen werden, um ein drahtloses Netzwerk bereitzustellen.

Die wichtigsten Netzwerk-Geräte sind also Switches und Router; auf wichtige Eigenschaften dieser Geräte wird in den Abschnitten *Switches* und *Router* näher eingegangen.

Das OSI-Modell

Damit Geräte miteinander kommunizieren können, müssen sie nicht nur physisch miteinander verbunden sein, sondern auch einige Regeln beispielsweise bezüglich der Daten-Formate und des zeitlichen Ablaufs der Daten-Übertragung einhalten. Derartige Regeln werden auch „Protokolle“ genannt; beispielsweise wird über Netzwerk-Protokolle (ähnlich einer Straßenverkehrs-Ordnung) festgelegt, wie Daten-Übertragungen in einem Computer-Netzwerk strukturell ablaufen müssen.

Das OSI-Modell untergliedert den Netzwerk-Verkehr in sieben verschiedene Ebenen („Layer“):

1. Ebene: „Physical“

   In dieser Ebene geht es darum, ob überhaupt ein physischer Kontakt zwischen verschiedenen Geräten besteht, ob diese also über Netzwerkkabel (oder eine Wireless-Verbindung) miteinander verbunden sind.

2. Ebene: „Data Link“

   Auf dieser Ebene arbeiten *Switches*: Sie können beispielsweise mehrere angeschlossene Geräte in logische Teil-Netzwerke untergliedern. Eine Kommunikation ist dann nur innerhalb dieser Teilnetzwerke möglich, jedoch nicht zwischen den einzelnen Teilnetzwerken.

3. Ebene: „Network“

   Auf dieser Ebene arbeiten Router: Hier werden unter anderem IP-Adressen, Subnetz-Masken, Gateways usw. festgelegt.

   Im Abschnitt *TCP/IP* ist diese Ebene näher beschrieben.

4. Ebene: „Transport“

   Auf dieser Ebene wird festgelegt, wie viel Daten auf einmal transportiert werden sollen.

Im Abschnitt *TCP/IP* ist auch diese Ebene näher beschrieben.

5. Ebene: „Session“

   Um auf einen Dienst eines (Web-)Servers zugreifen zu können, muss jedes mal eine „Session“ zu diesem aufgebaut werden. Im Session-Layer werden derartige „Sitzungen“ zu anderen Rechnern aufgebaut und verwaltet.

6. Ebene: „Presentation“

   Hiermit ist im Wesentlichen das Betriebsystem gemeint. Eine funktionierende Kommunikation setzt auf dieser Ebene beispielsweise voraus, dass die Geräte-Treiber richtig funktionieren und der Benutzer die entsprechenden Rechte besitzt.

7. Ebene: „Application“

   Auf dieser Ebene befinden sich Programme wie Firefox oder Thunderbird. Der Benutzer nutzt diese Ebene, um Daten mit anderen Rechnern auszutauschen.

Die Unterteilung der Netzwerk-Kommunikation in diese sieben Ebenen ist insbesondere dann nützlich, wenn eine solche *nicht* funktioniert, und man bei der Fehlersuche eingrenzen möchte, um welche Art von Fehler es sich wohl handelt. Hat beispielsweise das Netzwerk-Kabel einen Wackelkontakt, so muss nicht auf der Anwendungs-Ebene nach Bugs gesucht werden..

Um sich die Bezeichnungen der einzelnen Ebenen besser merken zu können, kann man folgenden Merksatz zur Hilfe nehmen:

Please	Physical
Do	Data Link
Not	Network
Tell	Transport
Secret	Session
Passwort	Presentation
Anyone	Application

Switches

Ein Switch ist ein Gerät, das mehreren Rechnern eines lokalen Netzwerks ermöglicht, miteinander zu kommunizieren. Ein Switch ist somit gewissermaßen das „Herzstück“ eines Netzwerks und sollte in seiner Bedeutung daher nicht unterschätzt werden.

Hubs und Switches

Um die Funktionsweise von Switches besser verstehen zu können, mag es sinnvoll sein, kurz auf die Vorgänger dieser Geräte einzugehen, nämlich den sogenannten „Hubs“.

Ein Hub ist/war ein Gerät, das ein einzelnes Netzwerk-Signal gleichmäßig auf mehrere Anschlüsse verteilen kann (ähnlich wie in einer Netzwerk-Dose das Leitungskabel beispielsweise über eine Wago-Klemme aufgeteilt werden kann). Dies hat zwar in der An-

fangszeit des Internets den Vorteil mit sich gebracht, auch mit zwei oder drei Geräten einen Internet-Zugang haben zu können und Computer miteinander kommunizieren zu lassen; diese Technik bringt allerdings gravierende Nachteile mit sich, weswegen Hubs heutzutage auf keinen Fall mehr verwendet werden sollten:

- Ein gravierender Nachteil von war/ist es, dass diese es nicht erlauben, eine exklusive Kommunikation zwischen nur zwei Rechnern aufzubauen, während noch weitere Computer mit dem gleichen Hub verbunden sind – auch bekommen ja stets alle Informationen mit.
- Ein weiterer gravierender Nachteil betrifft die Datenübertragung im Netzwerk. Im Ethernet werden Daten allgemein in Form von so genannten „Paketen“ übertragen. Soll beispielsweise eine Datei an einen anderen Computer geschickt werden, so wird dieses in tausende kleine Einzel-Pakete unterteilt; diese werden dann einzeln zum Ziel-Computer geschickt und dort wieder zusammengesetzt.

  Damit ein Computer Daten über einen Hub senden kann, darf dort nicht gleichzeitig ein anderer Daten-Verkehr vorliegen. Ein sendender Computer muss also warten, bis gerade kein sonstiger Datenstrom über den Hub fließt, und beginnt dann Pakete zu schicken. Dies wird beispielsweise dann zu einem Problem, wenn bei vier angeschlossenen Computern je zwei Computer „über Kreuz“ miteinander kommunizieren wollen. Senden zwei Computer gleichzeitig ein Paket, so kommt es zu einer Daten-Kollision. Die sendenden Computer warten dann einen zufällig langen Bruchteil einer Millisekunde, und beginnen dann erneut die einzelnen Datenpakete zu versenden.

Man kann sich leicht vorstellen, dass bei einer zunehmenden Zahl an angeschlossenen Geräten bei Hubs sehr schnell die Daten-Kollisionen überwiegen und der eigentliche Daten-Verkehr zum Erliegen kommt.

Switches arbeiten in vielerlei Hinsicht besser. Am wichtigsten ist wohl, dass sich diese Geräte „merken“, welches Gerät mit welchem Anschluss verbunden ist. Dieses „Gedächtnis“ funktioniert anhand der sogenannten „MAC-Adressen“: Jedes netzwerkfähige Gerät bekommt bei seiner Herstellung einen solche, weltweit einmalige Kennzeichnung. Sendet also ein Computer Daten an einen Switch, so merkt sich dieser automatisch die MAC-Adresse der entsprechenden Netzwerk-Karte. Die Daten werden dann nur einmalig an komplett unbekannte Geräte, ansonsten aber nur an den Zielrechner weitergeleitet, der wiederum über seine MAC-Adresse identifiziert ist. Eine gleichzeitige Kommunikation auch vieler angeschlossener Geräte ist somit problemlos möglich.

Wichtige Eigenschaften von Switches

Allgemein können sich Switch-Geräte die Identitäten der angeschlossenen Geräte merken. Darüber hinaus gibt es allerdings weitere Eigenschaften, bezüglich denen sich Switch-Geräte unterscheiden (und die meist auch einen preislichen Unterschied zur Folge haben):

- Managed/Unmanaged Switches:

  Die meisten Switch-Geräte für Privathaushalte sind sogenannte „Managed Switches“ – sie funktionieren automatisch, jedoch ohne weitere Konfigurations-Möglichkeit seitens des Anwenders.

Unmanaged Switches hingegen laufen zwar standardmäßig ebenfalls im „managed"-Modus, der Anwender hat allerdings die Möglichkeit, für jeden einzelnen Anschluss-Port individuelle Konfigurationen vorzunehmen.

- Geschwindigkeit:

  Alte Switches konnten Datenübertragungen von bis zu 10 Mb/s handhaben, neuere können auch Geschwindigkeiten von 100 Mb/s oder sogar > 1 Gb/s erreichen. Beachten sollte man allerdings, dass mit Mb Mega-Bit gemeint sind, nicht Mega-Byte: Ein Byte – die übliche Basis-Einheit für Datei-Größen auf Festplatten – besteht aus 8 Bits. Eine Geschwindigkeit von einem Giga-Bit je Sekunde entspricht somit einem möglichen Daten-Transfer von 125 Mega-Byte je Sekunde.

  Neben der Geschwindigkeit der einzelnen Ports ist auch von Bedeutung, wie groß der maximale Datenstrom ist, den das Gerät insgesamt (also über alle Anschluss-Ports hinweg) bereitstellen kann. Es ist beispielsweise möglich, dass auch ein 1 Gb/s-Switch mit 48 Ports insgesamt nur 32 Gb/s an Datenstrom umsetzen kann.

- „Trunk"-Anschlüsse:

  Switches mit vielen Ports haben oftmals zusätzlich zu den „normalen" Ports zwei sogenannte „Trunk"-Ports, die für eine Kommunikation zwischen mehreren Switch-Geräten vorgesehen sind. So ist es beispielsweise möglich, dass es in einer Firma einen Switch für jede Etage gibt. Möchte ein Computer mit einem Computer auf einer anderen Etage kommunizieren, so sendet der sich auf der gleichen Etage befindliche Switch das Signal an die übrigen Switches weiter, die wiederum prüfen, ob bei ihnen der Ziel-Computer angeschlossen ist.

- VLAN:

  Auf konfigurierbaren Switch-Geräten (unmanaged) sind so genannte „virtual local network areas" (VLANs) möglich: Die Anschlüsse eines solchen Switches können somit in mehrere logische Teil-Netzwerke untergliedert werden. Eine Kommunikation ist dann nur innerhalb dieser Teil-Netzwerke möglich, nicht jedoch zwischen den einzelnen Teil-Netzwerken. Dies kann beispielsweise von Bedeutung sein, wenn nicht nur Computer, sondern auch Telefon-Geräte (Voice over IP) an dem Switch angeschlossen sind.

- Power over Ethernet:

  Manche Switches bieten ein sogenanntes Power over Ethernet (PoE); damit können die Netzwerk-Kabel in gewissem Maß auch zur Spannungsversorgung der angeschlossenen Geräte verwendet werden. Möglich sind (ohne Gewähr) bis zu 24 V und $0,5$ A je Port, also maximal 12 W je angeschlossenem Gerät. Damit können beispielsweise Telefone, Wireless Access Points oder andere Switches beziehungsweise Router mit Spannung versorgt werden, auch wenn an dieser Stelle keine Steckdose verfügbar ist.

  Bietet ein Switch Power over Ethernet, so muss darauf geachtet werden, dass die auch mögliche Aufnahme-Leistung ausreichend groß ist. Möchte man beispielsweise an 10 Ports 12 W an Leistung über PoE für Endgeräte bereitstellen, so muss die Aufnahme-Leistung des Switches entsprechend auch mindestens 120 W betragen.

- Quality of Services:

Für verschiedene Geräte sind schnelle Datentransfer-Geschwindigkeiten unterschiedlich wichtig: Während Downloads von Emails oder System-Updates durchaus auch langsam sein dürfen, so sollte der Datentransfer für Live-Stream-Videos stets um einiges höher sein. Am wichtigsten sind schnelle Daten-Übertragungen allerdings für eine Echtzeit-Kommunikation, also beispielsweise Voice over IP.

Mittels konfigurierbaren Switches (unmanaged) können für die verschiedenen Arten von Daten-Paketen unterschiedliche Prioritäten festgelegt werden. Damit kann beispielsweise garantiert werden, dass die Daten-Übertragung von Echtzeit-Kommunikations-Geräten stets Vorrang hat.

Die grundlegenden Eigenschaften von Switches liegen im *OSI-Modell* auf Ebene 2 („Data Link“); bietet ein Switch auch eine Konfigurations-Möglichkeit für die Quality of Services, so entspricht dies im OSI-Modell bereits der Ebene 3.

Router

Router regeln die Kommunikation zwischen verschiedenen Teil-Netzwerken. Abgesehen von meist extrem teuren Geräten, die *ausschließlich* diese Aufgabe übernehmen (und somit auch nur zwei Anschluss-Buchsen haben), sind in Routern meist Switch-Geräte integriert, so dass Router meist eine Aufteilung eines Netzwerkes in verschiedene logisch getrennte Teil-Netzwerke ermöglichen.

Die meisten Router lassen sich über ein Web-Frontend konfigurieren, indem man in der Adress-Zeile eines Webbrowsers die Adresse des Routers eingibt.
Zu den wichtigsten Aufgaben eines Routers zählen folgende:

- DHCP:

  Das „Dynamic Host Configuration Protocol“ (DHCP) ermöglicht eine dynamische Zuweisung von IP-Adressen an individuelle, neu mit dem Router verbundene Geräte. Die Adressen für die neu angeschlossenen Geräte werden vom Router automatisch anhand eines „Address-Pools“ vergeben, der vom Benutzer festgelegt wird. Hat beispielsweise der Router die lokale Netzwerk-Adresse `192.168.1.1`, so könnten die Adressen von `192.168.1.100` bis `192.168.1.200` für DCHP freigegeben werden.

  Stationäre, für das Netzwerk wichtige Geräte wie Email-Server, Webserver, usw. hingegen sollten stets statische IP-Adressen haben, die allerdings auch manuell festgelegt werden müssen.

- DNS:

  Über das „Domain Name System“ (DNS) kann man Computer-Namen für bestimmte IP-Adressen zuweisen, beispielsweise `PC1` für `192.168.2.15`. Dies ist für Webadressen von großer Bedeutung, für lokale Netzwerke spielt diese Router-Fähigkeit hingegen meist nur eine untergeordnete Rolle.

- Port Forwarding

- Firewall

Exkurs: Netzwerk-Verkabelungen

Ein Netzwerk-Kabel besteht allgemein aus acht einzelnen Datenleitungen, wobei diese innerhalb des Kabels stets in vier Kabel-Paare untergliedert sind, die jeweils ineinander verdrillt sind („Twisted Pair“). Dies bewirkt eine bessere Abschirmung der übertragenen Signale gegenüber äußeren elektromagnetischen Störfeldern. Zusätzlich werden die vier Kabelpaare meist durch eine Kunststoff-Folie sowie ein dünnes Metall-Gitter abgeschirmt.

Die Abschirmung und somit auch die Qualität der Daten-Übertragung ist bei verschiedenen Netzwerk-Kabeln unterschiedlich gut.

- Die günstigste Variante, die auch heute noch oftmals verwendet wird, heißt CAT5 beziehungsweise CAT5e; falls solche Kabel verbaut werden sollen, so sollte zumindest CAT5e genutzt werden, um zumindest prinzipiell eine Daten-Übertragung im Gigabit-Bereich zu ermöglichen.
- Besser (und aus meiner Sicht empfehlenswert) sind CAT6-Kabel. In diesen befindet sich eine zusätzliche Abschirmung zwischen den einzelnen Kabel-Paaren, und diese haben einen größeren Leitungs-Querschnitt; es treten dadurch weniger Störsignale auf (auch zwischen den einzelnen Datenleitungen).
- Nochmals besser, aber auch wesentlich teurer, sind CAT7-Kabel. Diese haben einen nochmals größeren Querschnitt der einzelnen Kupfer-Leitungen, und eine nochmals bessere Abschirmung. Mit diesen Kabeln sind prinzipiell die zur Zeit höchsten Datenübertragungs-Raten möglich.

Die Geschwindigkeit, die bei der Datenübertragung in einem Netzwerk tatsächlich erreicht werden kann, wird durch das langsamste Gerät begrenzt. Es bringt beispielsweise nichts, CAT7-Kabel in Kombination mit einem „langsamen“ Switch oder Router zu verwenden. Gegenüber den einfachen CAT5-Kabeln haben die hochwertigeren CAT6- und CAT7-Kabel zudem einen etwas höheren Durchmesser, so dass man beim Anbringen von Steckern beziehungsweise Netzwerk-Dosen ebenfalls darauf achten sollte, dass diese für den jeweiligen Kabel-Typ geeignet sind (eine Abwärts-Kompatibilität ist stets gegeben).

Die maximale Länge eines Netzwerk-Kabels beträgt 100 m.

Netzwerk-Konfiguration mit `iproute2`

Das Programm-Paket `iproute2` zählt zum Standard-Umfang jedes modernen Linux-Systems; es umfasst die Programme `ip`, `ss` und `tc`.[3]

Eines der vielseitigsten Werkzeuge ist das Programm `ip`:

- Um alle Netzwerk-Geräte des eigenen Computers anzuzeigen, kann man folgendes eingeben:

[3] Die Programme des `iproute2`-Pakets können die auf alten Systemen häufig genutzten Programme `ifconfig`, `netstat`, `arp`, `route` und `brctl` vollkommen ersetzen; die zuletzt genannten sollten somit nicht mehr eingesetzt werden.

```
ip address show

# Ausgabe (Beispiel):

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 4096 qdisc noqueue state UNKNOWN group␣
↪default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state␣
↪UP group default qlen 1000
    link/ether 00:1e:06:33:31:a3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.103/24 brd 192.168.2.255 scope global dynamic eth0
       valid_lft 1732475sec preferred_lft 1732475sec
    inet6 fe80::21e:6ff:fe33:31a3/64 scope link
       valid_lft forever preferred_lft forever
```

Anstelle von `ip address show` kann kürzer auch `ip addr sh`, oder noch kürzer `ip a s` eingegeben werden: Da die Anweisungen hierdurch bereits eindeutig sind, werden sie von `ip` automatisch vervollständigt. Man kann sogar nur `ip a` angeben, da dann `show` als Standard-Anweisung genutzt wird.

- Eine kürzere Ausgabe liefert die Anweisung `ip link show` (beziehungsweise kürzer `ip l`:

```
ip link show

# Ausgabe (Beispiel):

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 4096 qdisc noqueue state UNKNOWN mode␣
↪DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state␣
↪UP mode DEFAULT group default qlen 1000
    link/ether 00:1e:06:33:31:a3 brd ff:ff:ff:ff:ff:ff
```

Mittels `ip link` lassen sich die Geräte nicht nur auflisten, es können vielmehr auch mittels `ip link set` ihre Eigenschaften geändert werden; insbesondere können einzelne Geräte damit aktiviert beziehungsweise deaktiviert werden:

```
# Deaktivieren beziehungsweise Aktivieren einer Netzwerk-Karte:
ip link set eth0 down
ip link set eth0 up

# Änderung des Interface-Namens:
ip link set eth0 name extern

# Setzen einer neuen MAC-Adresse:
ip link set eth0 address 00:80:41:ae:fd:7e
```

... to be continued ...

Netzwerk-Monitoring mit `tcpdump`

... to be continued ...

Firewalls mit `iptables`

... to be continued ...

Linux-Server

Konfigurations-Verwaltung mittels `ansible`

Virtualisierung von Betriebsystemen

...

... to be continued ...

Links

Linux-Distributionen

- Linux Mint (en.)
- Linuxmuster: Serverlösung für Schulnetzwerke (de.)

Linux (allgemein)

- Das Ubuntuusers-Wiki (de.)
- Das Linux-Kompendium (de, Wikibook)
- Die Selflinux-Dokumentation zu GNU/Linux (de.)
- Das Linuxwiki (de.)
- Anleitung zum Absichern von Debian (de.)
- Linux Online Book Collection (en.)
- Debian-Anwender-Handbuch

Shell und Shell-Scripting

Deutschsprachig:

- Shell-Infos und Befehlsübersicht (de.)
- Shellprogrammierungs-Tutorial (de.)
- Shell-Programmierung (de., Openbook)
- Shell-Programmierung (en.)
- Tips und Tricks zur Shell (de.)
- Advanced Shell Scripting Guide (en.)

Englischsprachig:

- Shell-Programm-Übersicht (en.)
- Link-Sammlung zur Shell (en.)

- Bash Guide for Beginners (en, PDF)
- Advanced Bash-Scripting (en, PDF)
- Bash Command List A-Z (en.)

Manuals:

- GAWK Programming (PDF, en.)
- GNU Make Manual (PDF, en.)

Quellen

Quellenangaben zu Shell und Shell-Scripting:

Die Beschreibung der Shell-Programme orientiert sich an *[Schaten2005]*; zusätzlich wurden die Manpages der einzelnen Programme genutzt.

Literaturverzeichnis

[Blum2011] Blum: Linux Command Line and Shell Scripting Bible. Wiley, 2011.

[Ploetner2011] Johannes Ploetner, Steffen Wendzel: Linux - Das umfassende Handbuch. Rheinwerk-Verlag Openbook, Bonn, 2011.

[Schaten2005] Ronald Schaten: Shell-Programmierung. Online-Skript (Creative Commons License), 2005.

[Wolf2005] Juergen Wolf: Shell Programmierung. GalileoPress, Bonn, 2005.

Stichwortverzeichnis

F

G

H

I

J

K

L

M

N

P

R

S

T

U

V

W

X

Z