Linux Kurs - v0.02

Franz Schäfer

2019-01-11

Contents

1 Wozu dieser Linux-Kurs?

Aus gutem Grund empfehle ich allen die mich auf Computer-Themen ansprechen die Installation von Linux. Wer allerdings noch nie damit gearbeitet hat, der/die muß relativ viel neues lernen. Das zahlt sich allerdings auch aus. Dieser Kurs ist für EinsteigerInnen gedacht die einen Überblick über Linux bekommen wollen. Neben rein technischen Themen gibt es dabei auch einen politischen Überblick über die Vorteile des Systems.

Bei der Auswahl der etwas näher Behandelten Programme habe ich auch darauf geachtet, was für uns politische AktivistInnen am ehesten relevant ist.

Angesichts Totalüberwachung und der Kontrolle der Geheimdienste durch Rechtsextremisten gibt es natürlich einen Schwerpunkt zum Thema Sicherheit und Verschlüsselung.

Ein Linux-System ist heute relativ einfach und schnell zu installieren und kann sehr weit rein auf der grafischen Oberfläche benutzt werden. Den echten Komfort eines Linux Systems merkt man/frau aber erst auf der Kommandozeile. In den weiter hinten liegenden Kapital geht das dann durchaus auch etwas in die Tiefe. Damit man/frau sich zu Helfen weiß, falls das System mal nicht so funktioniert wie es soll.

Die Aktuelle Version dieses Documents findest du auf `https://mond.at/lk`

2 Was ist "Linux"? Was ist "GNU"? Was ist "Freie Software"?

2.1 Was ist Linux?

Linux ist ein freies **Betriebssystem**. Ein Betriebssystem kümmert sich um den Zugriff auf die Hardware. Es verwaltet den Speicher, es "weiß" wie man Daten auf eine Festplatte schreibt, wie die Netzwerk-Karte angesprochen werden will, etc, etc.

Zum Betriebssystem gehört der so genannten "Kernel", der die oben erwähnten Aufgaben war nimmt und zusätzlich auch noch mehr oder weniger viele System und Anwenderprogramme die das ganze darum herum regeln. Z.b. Wie das verarbeiten eingegebener Befehle.

Das Betriebssystem bietet also Schnittstellen für andere Programme die dort laufen können (eine so genannte API) und eine Schnittstelle für die BenutzerInnen (z.b. Befehlseingabe und optional eine graphische Oberfläche).

Die Schnittstellen von Linux entsprechen denen von Unix. Vom GNU Projekt wurden viele Freie Unix Programme geschaffen auf denen die dann Linux aufbauen konnte. GNU ist eine rekursive Abkürzung für "GNU is Not Unix".

2.2 Was ist "Freie Software"? Was ist "Open Source"?

"Freie Software" und "Open Source" werden oft synonym benutzt. Richard Stallman, Vordenker der Bewegung bevorzugt den Begriff "Freie Software" um auf den politischen Aspekt hinzuweisen. "Open Source" wird i.a. von der libertären Seite benutzt: Die den Vorteil von Software, die mit Source Code, den man/frau auch selbst Verändern kann, zwar schätzen aber dies nicht als politisches Statement sehen wollen.

Als politisch aktive Menschen sollten wir daher eher den Begriff "Freie Software" verwenden, auch wenn der Begriff "Open Source" weiter verbreitet ist. Als Kompromiss bietet sich FLOSS an: "Free Libre Open Source Software".

Die Frage ist oft auch eine praktische: Wann ist Software "frei"?

Z.b: Manche Firmen geben ihre Software mit Source Code weiter, erlauben aber dessen Verwendung nur sehr eingeschränkt.

Eine praktische Definition zur Unterscheidung der sehr vielen Software Lizenzen wurde mit den "Debian Free Software Guidelines (DFSG)" geschaffen.

Für Richard M. Stallman (RMS) ist das "frei" wie in "Freiheit" und nicht wie in "Freibier" gemeint. Es geht um unsere Rechte.

Eine der wichtigsten Software Lizenzen ist die von Stallman geschaffene GPL.

3 Was sind die Vorteile von Linux und von "Freier Software"?

relativ zukunftssicher solange sich Menschen finden die ein wenig Ahnung von Software haben und die das Programmpacket nützen wollen, wird es das Packet weiter geben. Es kann nicht so einfach "vom Markt verschwinden".

erlerntes know-how bleibt weiter nützlich Wer ein Unix von vor 40 Jahren genutzt hat wird sich auch auf einem modernen Linux gut zurecht finden. Selbiges gilt auch für die meisten Programme. z.B.: LaTeX

alternative zum Kapitalismus für alle die auf ein relativ gut funktionierendes Beispiel von feier Kooperation verweisen wollen, ist Freie Software ein sehr gutes Beispiel.

Die UserInnen werden nicht eingesperrt Kommerzielle Programme zielen oft darauf ab, die AnwenderInnen daran zu hindern ihre Daten in andere Formate zu exportieren um diese an die eigene Software zu binden. ("Vendor Lock-In")

nicht nach Marketing Aspekten designed Kommerzielle Software, wird sehr oft nach Marketing Aspekten entwickelt. Es soll ein guter erster Eindruck entstehen damit die Software gekauft wird. Das danach ist dort oft nicht so wichtig.

keine Backdoors Da der Source Code offen ist und viele EntwicklerInnen an der Software arbeiten ist es viel schwieriger für Geheimdienste Überwachungshintertüren in Freie Software einzuschleusen.

bessere Qualität und Stabilität Obwohl es natürlich große Unterschiede in der Qualität von Software sowohl im kommerziellen als auch im freien Bereich gibt: generell ist Freie Software oft deutlich besser, schlanker, stabiler und sicherer. Zu Zeiten als Windows 3.1 AnwenderInnen noch mit täglichen oder stündlichen Abstürzen rechnen mussten, hatten Linux AnwenderInnen schon ein System das wochenlang ohne Probleme gelaufen ist.

software leicht zu installieren zumindest die (zigtausenden) Programme die Teil einer Linux Distribution sind, sind sehr leicht (Wie aus einem App-Store) zu installieren.

an eigene Bedürfnisse anpassbar . Notfalls mit Änderungen am Source-Code, sehr oft aber mit einfachen Skripts.

keine Abhängigkeit von Konzernen . Wer kauft die Firma als nächstes auf? Für uns Linux UserInnen nicht wirklich relevant.

Hilfe über Community Da das Programm selbst frei ist, sind auch viele AnwenderInnen eher bereit, gratis Hilfestellung für andere BenutzerInnen zu geben. Bei kommerziellen Programmen ist es eher so, dass die UserInnen denken: "Das soll doch der Hersteller machen."

Gibt es auch Nachteile? Ja: Manchmal ist freie Software nicht so "glattpoliert" wie manche kommerzielle Produkte. Für manche Aufgaben gibt es auch fast nur kommerzielle Software.

4 Lizenzen und die Politik Freier Software

4.1 Die 4 Freiheiten der GPL

Die GPL "General Public Licence" (Oft auch GNU Licence genannt, da von Stallman für die GNU Projekt entworfen) garantiert 4 Freiheiten:

run for any purpose Niemand darf einschränken wofür man/frau die Software verwenden will.

study and change Der Source Code muss dem Programm beigefügt werden oder einfach verfügbar gemacht werden damit Menschen aus dem Programm lernen können und sie es für sich verändern und erweitern können.

redistribute - help others Niemand darf die Weitergabe des Programmes und des Source Codes einschränken.

redistribute modified versions Niemand darf die Weitergabe von modifizierten Versionen des Programms einschränken. Einzige Einschränkung: Die modifizieten Versionen müssen Ebenfalls unter GPL verfügbar sein.

Vorteil der GPL ist, dass der Code der von der Community geschaffen wurde nicht von kommerziellen Konzernen missbraucht werden kann: Diese dürfen die Software zwar auch, wie alle Anderen, verwenden, müssen aber, wenn sie diese in Produkte einbauen, den Source Code auch unter GPL wieder weiter geben. Oft ist es für Firmen Sinnvoll das zu tun und damit vergrößert sich der Pool freier Software. (Die GPL ist "viral").

4.2 Andere Lizenzen; BSD, Creative Commons

Die wichtigste anderen freie Lizenz ist die BSD (bzw. BSD artige Lizenzen. z.b. MIT). Sie erlaubt mehr oder weniger "alles". Allerdings auch: die Weiterverwendung des Source Codes in kommerziellen Produkten bei denen der Code **nicht** weiter gegeben wird.

Firmen können BSD Lizensierte Freie Software daher leicht "stehlen". Microsoft und Apple haben sehr viel BSD lizensierten Code verwendet.

Die von Lawrence Lessig entwickelten CC ("Creative Commons") Linzenzen haben die GNU und die BSD Lizenzen als Vorbild. Die Vielfalt der Optionen bei CC bringt aber auch Probleme mit sich.

4.3 Die Politik Freier Software, Keimformen

4.3.1 Keimform

Der Erfolg freier Software in unserer Welt ist ein Beleg dafür, dass diese mit dem Kapitalismus bis zu einem gewissen Grad kompatibel ist. Firmen haben ein Interesse daran. Auch Firmen wie Google oder Apple wollen nicht von Microsoft abhängig sein. Firmen schätzen alles was Kosten senkt und wenn Freie Software das kann dann wird es auch eingesetzt.

Andererseits ist verwirklicht freie Software doch vieles von dem was wir als Kommunismus verstehen. Das Eigentum an Software wird, unter Ausnützung des Copyrights, aufgehoben und die Freiheit der Software wird damit geschützt. In Freie Software sehen wir die freie Kooperation freier Individuen - soweit diese unter den gegebenen Umständen möglich ist.

Man spricht daher auch von einer "Keimform". Freie Software zielt in Richtung "Überwindung des Kapitalismus", kann aber mit in ihm wachsen.

Ich denke das sollte für uns politische AktivistInnen durchaus Beispielcharakter haben: Wir müssen unsere Aktivitäten so gestalten, dass sie im hier und jetzt wachsen können und gleichzeitig auf die Veränderung eben dieser herrschenden Verhältnisse abzielen.

Zwei weitere Prinzipien aus der Produktion Freier Software die, meiner Meinung nach, für uns Relevant sind: "Forks" und "Show Running Code"

4.3.2 Fork

Ist jemand mit der Richtung in die sich ein Projekt entwickelt nicht einverstanden, so kann er oder sie das Projekt jederzeit "forken". D.h. man/frau macht Entwicklungen in eine andere Richtung. Wenn der Source Code dann auseinander läuft ist das zwar Ineffizient (und somit die Motivation hoch, einen Fork zu vermeiden und doch zusammen an einem Code-Strang zu Arbeiten hoch) aber immer eine Option. Die Freiheit getrennte Wege zu gehen besteht immer und wird ab und zu auch in Anspruch genommen.

4.3.3 Show Running Code

Oft meinen Menschen sie müssten mit guten Anregungen helfen. Wer aber wirklich etwas bewegen will sollte doch mit machen. Linus Torvalds, Ursprünglicher Entwickler von Linux, antwortet mit dem berühmten "Show running Code" einem anderen Entwickler der meinte man solle bestimmte Dinge ganz anders machen.

Hier also für uns als AktivistInnen: Wer es anders will muss auch mitmachen.

Einige dieser Überlegungen sind in den Artikel
`http://qummunismus.at/p/article120.html` eingeflossen.

Mehr Überlegenungen zur politischen Relevanz auch auf: `http://mond.at/texte/soak05_freie_software.html`

5 Commandozeile (CLI) versus Graphische Oberfläche (GUI)

Ein Linux kann heute durchaus gut ohne Kommandozeile benutzt werden. Für fast alles gibt es gute grafische Programme. Dennoch lohnt es sich auch in die Kommandozeile ("Command Line Interface (CLI)") einzusteigen. Eine gut gestaltetest CLI hat gegenüber einer grafischen Oberfläche durchaus viele Vorteile:

automatisierbar die befehle die man/frau händisch eintippt können sehr leicht auch in automatisch ablaufende Scripts eingebaut werden. Wenn z.b. eine Aktion jeden Tag zur selben Uhrzeit ausgeführt werden muss oder eine Aktion auf tausende oder Millionen Dateien angewendet werden muss und man/frau nicht alles händisch anklicken will.

leichter dokumentierbar, nachvollziebar wer schon einmal eine Anleitung für ein grafisches Programm durchgelesen hat: Das ist schwierig zu erstellen und schwierig zu erklären. Klicken sie auf das dritte Tab im Fenster und wählen sie den rot unterlegten Menüpunkt, etc. Dagegen ist es sehr einfach aufzuschreiben welche Befehle eingetippt werden müssen.

weniger resourcen aufwand Eine Grafische Oberfläche benötigt relativ viele Resroucen (Speicherplatz, CPU Leistung). Eine CLI kann auch auf extrem schwachen Geräten (z.b. Raspberry Pi) genutzt werden.

Fernzugriff leicht möglich auch die Benutzung über langsame Internet-Verbindungen hinweg ist mit CLI sehr bequem möglich.

Ausdrucksstärke . Hast du schon mal versucht dich alleine mit Gestik mit denen Mitmenschen zu verständigen. Die gesprochen und die geschriebene Sprache sind da deutlich Stärker in ihrer Ausdruckskraft. Ähnlich kann auch mit kurzen Befehlen viel mehr ausgedrückt werden wie mit langem herumgeklicke.

Hilfe Eine gute CLI bietet i.a. auch Hilfestellung an. Da die schriftlichen Befehle leichter dokumentiertbar sind, ist diese sogar oft besser als bei grafischen Programmen.

Natürlich hat auch eine GUI ihre Vorteile - insbesondere dort wo auch die Problemstellungen grafischer Natur sind: z.B. ein Zeichenprogramm.

6 Andere Betriebsysteme und gemeinsmae Nutzung mehrere Betriebsysteme

6.1 Einordnung

Der Kern von Linux ist der "Kernel". Der wird z.b. auch in **Android** (Googles Betriebsystem für Handys) verwendet. Allerdings hat Android dort nur Google-eigene Programme laufen. Viele davon ohne Source-Code, auch wenn Teile von Android auch unter einer relativ freien Lizenz verfügbar sind.

Der Linux Kernel und zum Teil auch Linux Programme werden in vielen **Appliances** (z.B. WLAN-Router, Alarmanlagen, NAS-Storage Server, etc, etc) eingesetzt. Dort hat man/frau als normale/r AnwenderIn meist keinen direkten Zugang zum System sondern nutzt das Ganze über ein Webinterface.

Üblicherweise meint man mit Linux aber ein mehr oder weniger vollständiges Linux (**GNU/Linux**) mit allen wichtigen Systemprogrammen. Man kann diese Programme auch immer selbst "compilieren" (d.g. vom Source Code in ein Programm umwandelt). Das ist jedoch aufwendig. Daher nimmt man i.a. fertige Distributionen in denn schon alle wichtigen Programme enthalten sind.

Mac OS/X ist ein kommerzielles Unix von Apple (das aus einem freien BSD Unix heraus entwickelt wurde aber jetzt nicht mehr frei ist). Wer sich auf der Linux Kommandozeile zuhause fühlt wird sich auch auf einem Mac mehr oder weniger zurecht finden. Die grafische Oberfläche ist aber nicht direkt mit der Unix X Oberfläche kompatibel. Viele freie Software und viele Linux Programme wurden auf Mac-OS/X portiert und können dort verwendet werden.

Obwohl Apple enorm von freier Software profitiert hat (Ohne den BSD Kernel wäre ihr hoffnungslos veraltetes und instabiles Mac-OS9 wohl das Ende von Apple am Desktop gewesen) ist Apple der Idee von Freier Software eher feindlich eingestellt.

Freie Programme können natürlich relativ leicht auf andere Systeme portiert werden. Viel der nützlichen Linux Programme wurden daher auch auf Microsoft-Windows portiert. Gimp, Inkscape, Blender, etc. können alle auch dort, mehr oder weniger gut, verwendet werden.

Chrome-OS Googles Chrom-OS ist ein schlankes Betriebsystem für Desktops. Es kommt ebenfalls mit einem Linux-Kernel ist aber kein volles Linux sondern nur als Unterlage für Googles Chrome-Web-Browser gedacht.

6.2 Windows und Linux gleichzeitig nützen

Wer längere Zeit mit Linux arbeitet will i.a. nicht mehr zurück. Gerade für den Umstieg am Anfang ist aber eine gleichzeitige Nutzung oft hilfreich. Am Ende ist es aber immer besser zu versuchen die Dinge die früher mit kommerziellen Programmen erledigt wurden mit Freien zu versuchen.

Dabei gibt es folgende Möglichkeiten:

dual boot Beide Systeme sind (entweder auf der selben, geteilten Festplatte oder auf verschiedenen Festplatten) am Computer installiert. Beim Booten wählt man mit einem Menü aus welches der Systeme gestartet werden soll. Die Nutzung ist zwar damit nicht ganz "gleichzeitig". Im Grunde aber die sauberste Variante. Jede der Systeme läuft ganz "native" und man hat am wenigsten Probleme mit Inkompatibilitäten.

virtualisierung Mit Virtualisierung wird ein kompletter eigener Computer "simuliert". D.h. Das System "denkt" es läuft auf einer eigenen Maschine, ist aber (wie bei "Matrix") in einer Simulation eines Computers gefangen. Wobei i.a. nur das originale Betriebssystem direkten Zugriff auf die Hardware des Computers hat (Netzwerk, Festplatte, USB, Grafik) und das simulierte System nur eine simuliete Hardware sieht. (Virtualisierungen gibt es auch für Mac OS/X).

Dabei gibt es zwei grundsätzlich verschiedene Möglichkeiten:

1. Mittels KVM im Linux einen Windows Computer simulieren. Das ist i.a. die bessere Variante: Damit ist der größte Schritt für den Umstieg auf Linux bereits vollzogen.
2. Für Virtualisierung gibt es im Windows mehrere Lösungen: das freie QEMU ist frei aber nicht all zu schnell. VMware, VirtualBox, Hyper-V sind kommerzielle Virtualisierungen auf Windows in denen man/frau z.b. einen virtuellen Linux Rechner betreiben kann.

Freie Programme unter Windows . Da Frei Software leicht zu portieren ist, gibt es viele Programme die ursprünglich für Linux entwickelt wurden auch im Windows. Gimp, Inkscape, Libre-Office, etc. Als Vorbereitung für einen später geplanten Umstieg ist das ein guter erster Schritt.

subsystem, API-emulation in Windows Das neue Windows 10 bietet eine Linux-Subsystem an. Die Programme laufen direkt im Windows benutzen aber API Schnittstellen die denen von Linux ähneln. (Ähnlich auch das freie CYGNUS Projekt, das versucht GNU Tools unter Windows zu betreiben). Das ganze ist nützlich wenn man aus irgend einem Grunde Windows benutzen muss, aber doch auch Linux und Unix Programme verwenden will.

wine Das Pendant dazu aus der anderen Richtung ist das **wine** Projekt, das versucht die Windows-API auch unter Linux zu Verfügung zu stellen. Das Windows **kein** freies System ist, und Microsoft diese Projekt natürlich in keinster Weise unterstützt, ist die Aufgabe schwierig. Wine ist daher alles andere als perfekt, hat aber inzwischen schon enorm viel geleistet. Viele Windows-Programme können so mehr oder weniger gut im Linux betrieben werden.

remote arbeiten: von Windows auf Linux und umkehrt wenn man 2 Computer hat dann kann der eine auf Linux und der andere Windows laufen und man kann übers Netzwerk Programme von einer Seite auf der anderen Seite benützen:

1. von Windows mit putty und SSH die Commandline im Linux benutzen. Braucht man nur die Linux Commandline, dann kann man/frau mit dem freien und sehr guten **Putty** auf ein entferntes Linux (auch am anderen Ende der Welt) einloggen und dort arbeiten.
2. Die grafische Oberfläche von Linux (X11) kann sehr leicht auf ein Windows umgeleitet werden. Dazu benötigt man eine X11 Emulation. z.b. Xming (frei) oder Exceed (teuer, kommerziell). Diese arbeiten auch gut mit putty zusammen, so dass die Verbindung durch eine Putty-SSH Verbindung getunnelt werden kann. Zuerst Xming starten und dann im Putty auswählen, dass die X11 Verbindung weitergeleitet werden soll.
3. mit dem **VNC** Protokoll kann sowohl eine Windows Oberfläche von Linux aus genutzt werden, als auch umgekehrt.
4. mit rdesktop und remmina kann man die in Windows eingebaute remote-Funktionalität (RDP) nutzen.

Die remote Funktionen sind natürlich auch für den Zugriff auf virtualisierte Systeme nützlich.

live CD siehe Unten. Zum "kurz einmal testen" sind natürlich Live-CD bzw Live-USB Sticks nützlich. Man kann in ein komplettes Linux booten ohne dies auf der Festplatte installieren zu müssen.

7 Unix/Linux Stammbaum: Die verschiedenen Distributionen

Neben Linux gibt es auch heute immer noch kommerzielle Unixe: z.B.: Solaris (oracle) oder IBMs AIX. Üblicherweise werden auch auf kommerziellen Unixen heutzutage viele GNU Tools installiert. Man/frau fühlt sich dort auch "zuhause".

Weiters gibt es viele freie, von BSD abstammende Unixe. z.B. FreeBSD oder OpenBSD. Im Vergleich zu Linux führen diese aber eher ein Nischendasein.

7.1 Distributionen

Da Linux frei ist kann jede/r kommen und es sich so einrichten und zusammenstellen wie er/sie möchte. Das ist viel Arbeit aber dennoch gibt es verschiedene Zusammenstellungen (Distributionen). Üblicherweise mit verschiedenen Schwerpunkten und Zielsetzungen.

Eines der wichtigeren Entscheidungskriterien ist, welches Packetmanagment System verwendet wird. Es gibt Debian (.deb) und RedHat (.rpm).

Manche Hersteller packen zu den freien Linux Programmen auch noch kommerzielle Programme dazu und/oder bieten kommerziellen Support an.

Hier eine unvollständige Auflistung wichtiger Distributionen

slackware - die erste Linux Distribution. Heute kaum noch verwendet

debian - die wichtigste freie Distribution. In der Kern-Distribution sind **nur freie** (im Sinne der DFSG - siehe oben) Programme enthalten. Man kann aber auch "non-free" dazu nehmen. Debian hat ein sehr großes Repertoire an Programmen und legt Wert auf einen möglichst einfachen und reibungslosen Update-Prozess. Alle 2 bis 3 Jahre gibt es eine neue "Major-Release" mit neuen Programmversionen. Dazwischen i.a. nur Security-Updates. D.h. das System ist sehr stabil und bleibt wie es ist.

ubuntu - kommerzielle Distribution. Für den Privatgebrauch Gratis. Basiert auf Debian. Gut für Einsteiger. Es gibt LTS ("Long-Term Stable") Versionen die ca. 4 Jahre lang gleich bleiben, aber security updates erhalten.

mint - auf debian und ubuntu aufbauende Distribution.

RedHat - kommerzielle Distribution, von IBM gekauft. Wird gerne im Firmenumfeld für Serversysteme genutzt (konservative Update-Politik, guter kommerzieller Support).

Fedora - gratis Version von RedHat in dem die allerneusten Features ausprobiert werden können.

CentOS - Freier RedHat Klone.

gentoo - Alle Packte werden i.a. immer direkt von Source erzeugt. (nicht wirklich für Einsteiger).

raspbian Debianvariante speziell für Raspberry-Pi

Was ist die besten Distribution für EinsteigerInnen? Ich empfehle Debian oder Ubuntu. Persönlich bevorzuge ich Debian.

7.2 Live CDs

Live-CDs sind CDs oder USB-Sticks, die direkt gestartet werden können, **ohne** dass dazu ein System installiert werden muss. Das ist praktisch fürs Ausprobieren aber auch nützlich für Troubleshooting (Wenn z.b. der Computer nicht mehr startet und man herausfinden will was das Problem ist), aber auch für Sicherheitsrelevante Operationen.

Hier 3 wichtige Live Systeme:

Knoppix Der Klassiker von Klaus Knopper. Basierend auf Debian (wobei meist auf die experimentelle unstable oder testing-Zweig gebaut wird). Fokus ist: "Linux ausprobieren".

Grml Ideal zum Trouble-Shooting. Das Schweizermesser für den Sysadmin.

Tails Fokus: Security und Verschlüsselung für Under-Cover Agenten. Die von Edward Snowden promotete Live-CD.

8 Linux auf der Grafischen Oberfläche (GUI)

Wir gehen hier davon aus, dass das System fertig aufgesetzt ist und zur Benutzung bereitsteht.

8.1 Wechsel zwischen Konsolen

Linux bietet mehrere "virtuelle" Bildschirme an, zwischen denen man/frau umschalten kann. Es gibt "grafische" Bildschirme und "Text Consolen".

Auf der Text Console steht meist irgend etwas von "tty" und "login:". Auf der grafischen Oberfläche gibt es ein Bild mit einer Anmeldemaske.

Umschalten kann man (wenn das nicht explizit ausgeschalten ist) mit: **Ctrl-Alt-F1** bis Ctrl-Alt-F9[1]. D.h. Die Strg Taste gemeinsam mit der linken Alt Taste (i.a. links neben der Leertaste) und gemeinsam mit der Funktionstaste F1 drücken.[2]

Wichtige andere Tastenkombinationen:

Ctrl-Alt-Backspace [3]. Killed die grafische Oberfläche (die Taste ist aber oft ist deaktiviert).

Ctrl-Alt-Delete [4] Startet den Computer (sauber) neu. Ist aber meist nur in der Text-Console erlaubt.

[1] Auf deutschen Tastaturen ist Ctrl mit **Strg** beschriftet

[2] Auf Laptop Tastaturen muss zum benutzen der Funktionstasten manchmal eine spezielle Umschalt-Taste benutzt werden.

[3] Backspace ist die Pfeil-nach-Links Taste rechts oben

[4] Delete ist auf deutschen Tastaturen mit "Entf" beschriftet

8.2 Elemente der GUI im Linux

Basis ist ein Programm das den Bildschirm verwaltet. Meist Xserver oder X11 genannt. Manche Distributionen setzen auch auf ein neueres System namens "Wayland".

Darauf läuft zunächst ein Programm zum einloggen. Üblicherweise kann dort auch ausgewählt werden welche der verschiedenen grafischen Oberflächen man/frau benutzen will. Es können auch mehrere BenutzerInnen gleichzeitig eingeloggt sein und man/frau kann zwischen diesen wechseln.

Um eine bestimmte Oberfläche auszuwählen muss **vor** dem Einloggen das Settings/Zahnrad Symbol geklickt werden.

Welchen man/frau benutzen will ist Geschmackssache. Am besten einmal ausprobieren:

8.3 Desktop-Environments und Window-Manager

Ein Windowmanager ermöglicht das Umschalten zwischen Programmen, Vergrössern, Verkleinern, etc. Ein Desktop-Environment kommt zusätzlich mit Funktionen für Datei-Managment und Weitere Funktionen.

GNOME ist ein kompletter Desktop Manager bietet aber auch eigens Window-Management. GNOME ist heute der Standard am Linux-Desktop. Man kann ihn auf verschiedene Arten nutzen. Ich bevorzuge die "Klassik" Variante.

KDE ist eine alternative Desktop Umgebung.

xfce Fenster-Manager/Desktop Manager mit sehr klassischem Look-And-Feel

lxde sehr kleiner, leichter Fenster Manager

andere fluxbox, i3, IceWM (win95 look), Enlightenment, ...

Üblicherweise ist es kein Problem, z.B.: für KDE Entwickelte Programme im GNOME zu benutzen und umgekehrt.

Alle BenutzerInnen können sich eigene Desktop Umgebungen aussuchen, bevorzugte Sprache und sogar andere Zeitzonen.

9 Passwörter

Am Beginn der Linux Benutzung steht die Wahl eines Passwortes. Extrem wichtig ist es hier ein gutes Passwort zu wählen. (Linux bietet viele Möglichkeiten der Remote-Nutzung und damit kann ein Linux Computer mit einem schlechten Passwort sehr schnell gehackt werden. Auch wenn man/frau initial keine Remote-Zugriffe einplant: besser gleich ein gutes Passwort wählen:

- Mindestens 8 Zeichen. Besser 9 oder 10.
- ein Mix aus Buchstaben, Ziffern und Sonderzeichen - idealerweise die Ziffern nicht nur am Ende sondern auch mitten im Wort.
- kein Wort das im Wörterbuch irgend einer Sprache steht oder auch nur entfernt Ähnlichkeit mit einem Wort aus einem Wörterbuch hat. (In einem typischen Wörterbuch stehen etwa 30000 Wörter. Ein Hacker der 10000 Passwörter pro Sekunde ausprobieren kann hat das Wort in 3 Sekunden gefunden!)
- idealerweise sollten alle Systeme verschiedene Passwörter haben. Keinesfalls ein Passwort verwenden, dass auch bei fremden Online-Serivces benutzt wurde. [5]
- Passwörter nicht online oder in Dateien ablegen und **niemals** über unsichere Kanäle wie SMS oder E-Mail versenden.
- oft wird empfohlen Passwörter öfter zu ändern. Das ist gut. Wichtiger ist aber die Befolgung obiger regeln.

[5] Für Online-Services einen Passwortmanager benutzen, Siehe: Kapitel 20.9, Seite 34

10 Arbeiten am gnome desktop

- Programme starten - über Startemenü und Shortcuts
- Virtuelle Workspaces - Linux bietet meist mehrere getrennte Arbeitsbereiche (unabhängig von den oben beschriebenen virtuellen Konsolen)
- cut and paste im X11 (Linke Maustaste: markieren und kopieren, mittlere Maustaste: Fallen lassen). Alternativ auch klassisches Ctrl-C Ctrl-V in vielen Programmen.
- WLAN Login - Üblicherweise lassen sich elementare Netzwerkeinstellungen und ein WLAN-Login direkt am Desktop erledigen. (Icon)
- ein Terminal starten
- umschalten zwischen Programmen - i.a. wie im Windows mit: Alt-Tabulator. Alternative Einstellungen mit Maus-Fokus sind möglich.
- Ein normaler Benutzer darf üblicherweise keine globalen Systemeinstlungen ändern. Dafür ist ein extra Passwort/ein Benutzerwechsel notwendig.
- den grafische Packetmanager benutzen.
- Packetmanagement auf der CLI

```
$ sudo su -
# apt-get update
# apt-get upgrade
# apt-get install programmname
# apt-cache search suchbegriff
```

`sudo su` - macht eine/n BenutzerIn zum root. `apt-get update` - holt sich eine aktuellisierte liste der verfügbaren Packete einer debian basierenden Distribution. `apt-get upgrade` - erneuert alle Packte (meist Sicherheitsupdates). `apt-get install programmname` - installiert ein neues Programm mit genau diesem namen. `apt-cache search nach-irgenwas` - sucht nach-irgendwas in der liste der Programmpackete - falls der genaue Name des Programms nicht bekannt ist.

11 Web Browser

Wichtigstes Werkzeug für fast alles heutzutage ist der Web-Browser. Folgendes steht zur Auswahl:

firefox - ist der Standard für freie Webbrowser.

google-chrome - ist der kommerzielle Webbrowser von Google. Dafür aber immer mit den aktuellsten Features. Als Zweitbrowser gut - falls im Firefox mal eine Seite nicht so gut funktioniert.

chromium - die open-source Variante von Googles Chrome Browser.

konqueror - KDE eigener Webbroser und Filemanager.

w3m - ein Webbrowser der auf der text-console funktioniert.

curl, wget curl und wget sind Tools die auf der Text-Console verwendet werden können um Dateien aus dem Web herunter zu laden.

z.B.: `wget https://mond.at/cd/txt/0001-ssh.txt`

Sehr leicht lassen sich damit auch ganze Websites spiegeln.

12 File Browser und das Linux Filesystem Layout

Alle Desktop Umgebungen kommen auch mit grafischen Datei-Managern. Egal ob man/frau den grafischen Datei-Manager benutzt oder auf ob auf der Text-Konsole gearbeitet wird, brauchen wir einen Überblick wo Dinge abgespeichert werden.

Im Unterschied zu Windows und DOS werden im Linux, so wie in anderen Unixen auch, Verzeichnisse und Dateinamen mit einem Schrägstrich nach vorne "/" getrennt. Im Unterschied zu DOS und Windows gibt es **keine** Laufwerksbuchstaben. Andere Laufwerke werden als so genannte "mounts" an beliebiger Stelle im Baum eingehängt. Was im Windows **C:**
ist, ist im Linux bloß **/** - d.h. das so genannte "root Verzeichnis" - die Wurzel des Baumes. Wie im Windows ist punkt-punkt **..** ein Wechsel auf das darüberliegende Verzeichnis. Ein Punkt alleine: **.** meint immer das aktuelle Verzeichnis. Beginnt ein Verzeichnispfad mit einem / so ist dies ein "absoluter Pfad" - d.h. vom Root-Verzeichnis weg. Ohne diesen / am Anfang ist es ein "relativer Pfad" - d.h.: vom aktuellen Verzeichnis ausgehend.

Ein weiterer, wichtiger Unterschied zu Windows: Linux ist "case-sensitive". Groß- und Kleinschreibung werden unterschieden. bild.jpg und Bild.jpg sind zwei verschiedene Dateien.

Files und Verzeichnisse die mit einem Punkt beginnen, sind so genannte "hidden files" und werden normalerweise nicht angezeigt.[6]

Eine Tilde-Slash **~/** ist oft ein Kürzel für das eigene Home-Verzeichnis.

Umlaute, Leerzeichen und Sonderzeichen sollten in Datei- und Verzeichnisnamen vermieden werden. Minus und Underline sind aber Ok. Es funktionieren auch alle anderen Zeichen (ausser /) aber man kann sich damit verschiedene Probleme einfangen. (Z.b. beim Austausch mit anderen Betriebssystemen oder beim Eintippen auf der CLI).

12.1 Die wichtigste Verzeichnisse

/ - Das Root-Verzeichnis - die Wurzel des Baumes.

/home - Unterhalb von /home haben alle User ein persönliches Home-Verzeichnis für die eigenen Dateien. (Ausgenommen: der root user). Z.B.: der Benutzer "karl" hat sein Home-Verzeichnis in **/home/karl**

/tmp - dort liegen "temporärer" Dateien. Das Verzeichnis wird bei einem Neustart meist gelöscht.

/usr /sbin/ /bin - In diesen Verzeichnissen liegen die meisten installierten Programme.

/etc - Dort liegen alle globalen Einstellungen für den Computer - üblicherweise dürfen diese von einem normalen User nicht verändert werden.

/mnt oder /media - dorthin werden üblicherweise Wechselmedien (USB-Stick, CD-ROM) "gemountet" - d.h. als Laufwerk eingebunden.

/proc und /sys - sehen aus wie echte Dateien, sind aber in Wirklichkeit nur ein Blick auf interne Einstellungen des Linux-Kernels. z.B. auf **/proc/cpuinfo** kann man sehen welche CPU bzw CPU-Cores der Computer hat.

/root - Das Homeverzeichnis für den root-Benutzer (Superuser).

Im eignen Home-Verzeichnis kann man/frau sich frei bewegen. Übliche Konventionen sind aber: `Desktop/` entählt Dateien die am Desktop angezeigt werden. `Downloads/` für Files die im Web-Browser heruntergeladen wurden, etc.

12.2 Den Verzeichnisbaum auf der Kommandozeile erforschen

In einer Text-Console wird man/frau üblicherweise mit einem ''Prompt" begrüßt. Der könnte z.b. So aussehen:
`karli@meinlaptop:~$`

Mit der Eingabetaste bekommt man einen neuen Prompt. Mit Pfeilnachoben oder Ctrl-P kann man in den Befehlen zurückblättern.

Ctrl-K löscht alles bis zu Ende der Zeile.

Wichtige Befehle um sich im Verzeichnisbaum zu bewegen:

pwd - "print working directory" zeigt das aktuelle Verzeichnis. Das ist meist aber auch im Prompt zu sehen.

ls - zeigt alle Dateien und Verzeichnisse im aktuellen Verzeichnis (ausgenommen "hidden files").

[6] Im grafischen Datei-Manager Ctrl-H drücken um Hidden Files anzuzeigen. Auf der Konsole: `ls -la`

ls -l - zusätzliche Infos zu den Files (z.b. Datum, Zugriffsrechte, Größe, etc.)

ls -a - auch "hidden files".

ls -ltr - nach zeit (**t**ime) geordnet. neueste zuletzt (**r**verse).

cd - wechsel (Current Directory) ins Home-Verzeichnis.

cd /home/karli - Wechsel zum einem absoluten Pfad. Hier /home/karli. Bei der Eingabe mittels der Shell kann die Tabulator Taste genutzt werden um die Eingabe zu Vervollständigen. Also: cd /h**TAB**/k**TAB**. Ist die Vervollständigung nicht eindeutig (z.B. es gibt auch eine Benutzerin katja mit einem /home/katja Verzeichnis) so wird das erste TAB ignoriert. Ein zweites mal TAB drücken zeigt dann die möglichen Vervollständigungen an. Mittels TAB-Completion ist ein extrem schnelles und bequemes Arbeiten auf der CLI möglich.

cd ~ ~wechselst auch ins eigene Homeverzeichnis.

cd . Punkt ist eine Abkürzung für das aktulle Verzeichnis. Wir bleiben wo wir sind.

cd .. eine Ebene höher. (Der Unix-Baum steht am Kopf und wächst nach unten).

mkdir test123 legt ein neues Verzeichnis namens test123 an. Vorausgesetzt wir dürfen das im aktuellen Verzeichnis.

cd test123 wechselt in das neu angelegte Verzeichnis.

touch bla.txt legt ein neues, leeres file namens [bla.txt] an. Falls bla.txt schon existiert hat bekommt es nur einen neuen Zeitstempel.

df zeigt alle mounts (Laufwerke) an, wo sie eingehängt sind und wieviel Platz (in kilobyte) noch ist.

df -h hhuman readable. Der Platz wird in G und M und K Bytes angezeigt.

du -hs Dokumente/ Disk-Usage: zeigt (-h "human readable.") den Verbrauchten Speicherplatz des Dokumente/ Ordners an.

mount es werden auch interne system-mounts angezeigt und mehr technische Details zu den einzelnen Dateisystmen.

find . es werden alle Files und Verzeichnisse vom aktuellen (.) Verzeichnis aus aufgelistet (Abbruch mit Ctrl-C).

13 Office Programme

Wer wie aus der Windowswelt gewohnt arbeiten will oder muss:

libreoffice Ein ziemlich komplettes und relativ Microsoft-kompatibles Office Packet. Mit Writer, Calc (Spreadsheet), Impress (Präsentationen), Base (Datenbanken - ähnlich wie MS-Access).

gnumeric Ein leichtes Spreadsheets aus der GNOME Familie

abiword, Ein leichtes Textverarbeitunsprogramm

Alternativ kann man natürlich auch Texte mit LaTeXverarbeiten und anstatt Berechnungen in einem Spreadsheet zu machen kann man echte Mathematik Software verwenden. Z.B. Octave.

14 Text Editor

Sehr viele Funktionen im Linux werden über normale, einfache Text Files abgewickelt. Diese enthalten keine Formatierung sonder nur reinen Text.

Konfigurationsdateien sind fast immer reine Text-Files. Skripte und Programmiersprachen verwenden reine Text-Files. LaTeXist reiner Text. etc.

Sich gut in einem Text Editor zurecht zu finden ist eine der wichtigsten Skills die man/frau braucht um gut mit Linux zurecht zu kommen.

Idealerweise auch einen Text-Editor der alleine auf der Kommandozeile, also auch ohne grafische Oberfläche funktioniert.

Hier eine Übersicht über einige der gebräuchlichen Text Editoren:

gedit ist der grafische Text Editor von GNOME. Er funktioniert daher nur auf der grafischen Oberfläche. Kann aber Syntax-Highlighting und Rechtschreib-Unterwellung.

nano ist ein sehr kleiner, sehr einfacher Text Editor ganz ohne jeden Komfort. Falls man/frau sonst noch keinen gelernt hat und schnell einen benötigt. Wichtige Tastenkürzel:

Ctrl-G Hilfe ein/Hilfe aus.

Ctrl-X Beenden. (mit Y Speichern mit N Nicht speichern.)

Ctrl-O Speichern. (Fragt nach dem Filenamen).

joe ist eher unüblich unter Unix Usern aber ich verwende ihn gerne. Sehr schlank, hat ein Word-Star-Kompatibles Tastaturlayout und kann Syntax-Coloring.

vi bzw vim (Sprich: Wie-Ei) ist der Standard-Editor für Unix. Er ist extrem leistungsfähig aber nicht ganz leicht zu lernen. Basis vi-Kenntnisse sollte man aber auf jeden Fall haben, da es oft der default Editor ist der verwendet wird. Wer bis jetzt noch keinen Text Editor genauer erlernt hat: Es zahlt sich aus VI zu lernen.[7] VI hat 3 Modi:

1. Befehls- und Navigationsmodus (in diesem Modus kommt man nach dem Start)
2. Text Eingabemodus
3. : Kommandozeilenmodus

ESC Aus dem Eingabemodus in den Befehlsmodus wechseln

i Insert - In den Texteingabemodus wechseln. (ESC zum verlassen des Modus)

: zum Beginn einer Befehlszeile (Kommandozeilen Modus - mit der Eingabetaste Beenden).

:w EINGABE speichern

:w blabla.txt EINGABE unter neuem Namen blabla.txt speichern

:q! Editor beenden ohne zu speichern. (alternativ zweimal groß-Z: **ZZ**)

:wq EINGABE beenden und speichern.

j hinunter (alternativ: Cursor Taste)

k hinauf (alternativ: Cursor Taste)

h rechts (alternativ: Cursor Taste)

l links (alternativ: Cursor Taste)

x Zeichen unter Cursor löschen

5x die nächsten 5 Zeichen löschen

dd aktuelle Zeile löschen

p zuletzt gelöschte Zeile wieder einfügen.

:help hilfe. mit **:quit** die Hilfe wieder beenden.

emacs bzw. Emacs-ähnliche Editoren wie **jed** (simple). Emacs ist ein extrem leistungsfähiger, in Lisp Programmierbarer Editor mit sehr vielen Funktionen. Inklusive Psychologe und Web-Browser.

Ctrl-x Ctrl-c Beenden und speichern oder nicht speichern (y oder n)

Ctrl-x s Speichern

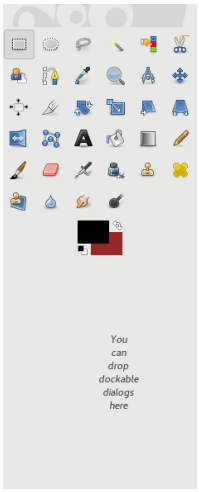
Ctrl-x Ctrl-w Unter anderem Namen speichern.

Wie gesagt: Es reicht einen der Editoren gut zu beherrschen. Wer noch keinen der Editoren gelernt hat kann sich mit nano oder gedit durch schummeln.

15 Bildbearbeitung mit Gimp & Co.

[7] Übersicht: `https://de.wikipedia.org/wiki/Vi` Tutorial z.B.: `https://de.wikibooks.org/wiki/Learning_the_vi_editor`

Gimp ist ein sehr leistungsfähiges Bildbearbeitungsprogramm. Es dient zur Bearbeitung von Pixel-Grafiken und Fotos.



- für die wichtigsten Funktionen ist die "Toolbox" nützlich. Das Toolbox-Fenster kann mit Ctrl-B angezeigt werden. (Oder über das Menü: Windows/Toolbox)
- mit Plus und Minus Tasten kann hinein/heraus-gezoomt werden.
- ein klick auf den Pinsel zum malen. Doppelklick auf den Pinsel um die Größe und Form des Pinsels zu verändern. Das Farbkästchen erlaubt es die Zeichen- bzw Hintergrundfarbe zu verändern.[8].
- Ctrl-Z zum Rückgängig machen.
- beim Speichern will Gimp im Gimp eigenen xcf Format speichern. (Damit werden alle Einstellungen und Layer und Selektionen mit gespeichert). Üblicherweise will man aber sein Bild nur einfach in ein gängiges Format (jpb, png, gif, ...) exportieren.
- Wichtig zum Bearbeiten sind die Selektierfunktionen.
  - mit shift-Taste halten: hinzu-selektieren
  - mit ctrl-Taste halten: weg-selektieren
  - mit Rechteck und Kreis/Ellipse entsprechende Bereiche selektieren
  - mit Freihand oder Bezierkurven selektieren
  - gleiche Farbbereiche selektieren
  - mit der Schere "intelligent" selektieren
  - Selektionen Vergrößern und Verkleinern
  - Selektion umkehren
- Ebenfalls wichtig für komplexere Aufgaben sind Layer Ctrl-L
- Eine große Palette an Filtern und Tools stellt enorm viele Funktionen bereit.
- Farbkurven sind sehr nützlich
- Die Bearbeitung kann in verschiedenen Modi erfolgen: Index (z.b. für GIF), RGB und Graustufen.
- Bilder können skaliert werden: Angabe in Pixel, Prozent oder in Maßeinheiten (Abhähngig von der frei wählbaren Auflösung).
- Die Größe der Zeichenfläche kann verändert werden.

Neben Gimp gibt es noch sehr viele anderen Bildbearbeitungstools. z.B: kann man mit dem befehl **import bla.png** einen Screenshot machen und in die Datei bla.png speichern. import ist ein Tool aus dem ImageMagick Serie. Damit lassen sich auf der Kommandozeile über 200 Bildformate konvertieren.

feh und **xli** sind schnelle Bildbetrachter.

15.1 Vektor Grafik

Auch zur Bearbeitung von Vektor Grafik gibt es viele Tools im Linux. Eines der Besten ist **inkscape**.

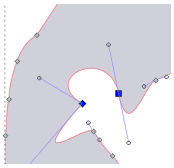
Das Inkscape eigene Format ist **svg**, das auch im Web inzwischen sehr gut unterstützt ist. Es lassen sich aber auch anderen Formate lesen und speichern. Auch das Konvertieren von und in Pixel-Formate ist möglich.

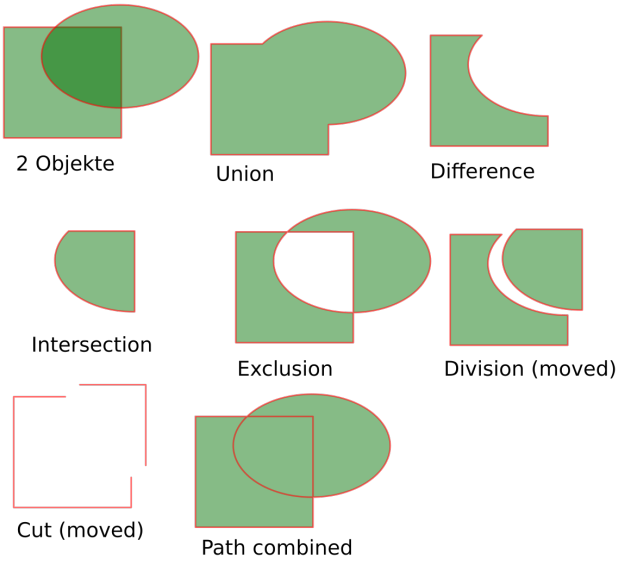
Hier ein kleiner Überblick über einige der wichtigsten Funktionen von inkscape:

- mit dem **freehand** Tool lassen sich offene und geschlossene Kurven direkt zeichnen.
- mit dem **bezier** Tool lassen sich eckige und Bezier-Kurven zeichnen. Bezierkurven sind schön gerundet. Die Rundungen lassen sich mittels verschiebbarer Tangenten gestalten.
- Wie im Gimp funktionieren + und - Tasten für den Zoom. Da in Vektorgrafiken keine Pixel gespeichert sind kann man/frau fast unendlich zoomen.

[8] die Hintergrundfarbe ist vor allem für den Radierer wichtig

- Mit dem Pointer-Tool und anklicken oder rechteckiger Selektion werden Objekte ausgwählt
- Objekte können verschoben, skaliert und gedreht werden (Der Drehpunkt kann verschoben werden).
- Über den “Fill and Stroke” Dialog wird die Farbe der Füllung und die der Striche ausgewählt. (CMYK Unterstützung für Hochwertigen Druck).
- die Eck-Punkte der Polygone bzw die Stützpunkte der Bezierkurven können nachträglich mit den “node editor” bearbeitet werden.
  - die Tangenten können verschoben werden um die Rundungen zu Verändern. Knoten können symmetrisch sein: Dann sind die Tangenten in beide Richtungen gleich.

    
  - nodes können gelöscht werden
  - neue nodes können zwischen 2 anderen nodes eingefügt werden.
  - Kurven können geschlossen werden oder aufgetrennt.
  - **Ctrl-L** vereinfacht kurven mit zuvielen knoten.
- Objekte können zu Gruppen zusammengefasst werden. (Achtung manche Funktionen können nur auf einzelne Objekte/Pfade angewendet werden und nicht auf ganze Gruppen.
- Pfade können mit logischen Operationen verknüpft werden (Und, Oder, Exclusive, Difference)
- Pfade können durch andere geteilt werden.



- Pfadeefekte (z.b: Koch, Rotate Copies)
- Bitmap Grafiken können in Vector Umgewandelt werden
- Vekotr Grafiken können als Bitmap exportier werden

16 Weitere Nützliche Software

16.1 Desktop Publishing - Scribus

Desktop Publishing. Wie Adobe-Indesign, nur Frei. Wenn mehrere AktivistInnen gemeinsam eine kleine Zeitung (Poldi) oder einen Folder herausgeben wollen.

16.2 3D Animation: Blender, Openscad

Blender ist ein extrem Leistungsfähiges 3D Programm. Es wurde für 3D-Animationen entwickelt und wird auch in professionellen Studios eingesetzt um Grafik-Effekte für Filme zu produzieren. Kann auch zum Filmschnitt verwendet werden. Aufgrund der vielen Funktionen gibt es aber einen hohen Lernaufwand. Ich setzte es auch ein um Modelle für 3-Druck zu erzeugen.

openscad ist ebenfalls zum Erzeugen von 3D-Strukturen, allerings mehr als Programm. Damit lassen sich parametrische 3D Objekte erzeugen und ist damit in der 3D-Drucker Community sehr beliebt.[9]

16.3 Audio und Video Bearbeitung: audacity, kino, LMMS

audacity bietet viele Funktionen zur Audio Bearbeitung.

Kino ist sehr gut für einfachen Filmschnitt geeignet. (Auch **blender** (Siehe oben) eignet sich sehr gut für Videoschnitt.)

LMMS ein Musiksynthisizer der alle Stückerl spielt.

16.4 PDF

evince ein guter PDF-Viewer, mit Suchfunktion.

pdftk Commandline Tools zur pdf Bearbeitung. Unter kann es:

- Seiten herausfiltern
- Mehrere PDK zu einem zusammenfügen
- Wasserzeichen einfügen

```
pdftk file1.pdf file2.pdf anhaenge*.pdf ouput ergebnis.pdf
pdftk original.pdf cat 17 155-158 2 320-end ouput auswahl.pdf
```

pdftotext kann den Text aus einem PDF herausholen. (allerdings nur wenn der Text im PDF vorhanden ist. Mit einem PDF das nur aus einem Bild besteht geht das nicht. Hier kann aber ein OCR Tool helfen: **tesseract**

pdfjam und **pdfjoin** sind ebenfalls nützlich. (Ähnlich pdftk).

16.5 LaTeX

Wer eine professionelle LaTeX- Umgebung benötigt ist mit Linux gut bedient. Da ist eine komplette latex-Installation mit allen Tools enthalten.

16.6 Mathematik

octave ein Klon der kommerziellen MatLab Umgebung. Ideal zum für Berechnungen aller Art.

sympy symbolische Mathematik

R "das" Statistikpaket

16.7 Programmierung

gcc, perl, python, ...- Die meisten Programmiersprachen sind gut in Linux integriert.

[9] Ein von mir erstelltes, parametrisierbares Objekt: `https://www.thingiverse.com/thing:3012219`

17 CLI basics

17.1 Der BASH Prompt

```
karli@meinlaptop:~$
```

In einer Text-Console wird man/frau üblicherweise mit einem ''Prompt" begrüßt. Der könnte z.b. So aussehen:

Mit der Eingabetaste bekommt man einen neuen Prompt. Der Prompt lässt sich konfigurieren. Üblicherweise ist er so eingestellt dass wir sehen unter welchem User wir arbeiten, auf welchem System wir arbeiten (könnte auch eine sein, dass am anderen Ende der Welt steht) und das aktuelle Verzeichnis. ein $ sagt uns dass wir "normaler User" sind. Ein # ist i.a. für den root User.

Es gibt verschiedene Shell Programme die uns die Kommando-Eingabe ermöglichen. Default im Linux ist die **bash**. Alternativen sind **tcsh** (default bei Mac OS/X). Beliebt ist auch die **zsh**.

17.2 Bewegen im Verzeichnisbaum

Mit den Befehlen `pwd, cd, df, ls ls -l, ls -ltr` (siehe voriges Kapital) können wir uns im Verzeichnisbaum bewegen und uns Dateien und Verzeichnisse ansehen.

17.3 keyboard shortcuts für die BASH

ctrl-p ctrl-n cursor up/down

history der history Befehl zeigt die zuletzt eingegebenen Befehle.

ctrl-c Abbruch des aktuellen Befehls.

ctrl-a zum Zeilenanfang

ctrl-k löschen bis zum Ende der Zeile

ctrl-r "**r**everse incremental search" - suche in der history durch Eingabe von Buchstaben.

ctrl-s stoppt die Ausgabe am Terminal.

ctrl-q Ausgabe wieder erlauben. (Continue).

ctrl-z stoppt den gerade laufenden Befehl. Durch Eingabe von **bg** wird dieser in Hintergrund geschickt und läuft dort weiter. Damit ist die shell wieder frei zur Eingabe weiterer Befehle. Will man/frau einen Befehl gleich im Hintergrund starten geht das mit einem & am Ende des Befehls. z.B.: `sleep 30 &`. Mit **fg** kann ein im Hintergrund laufender Prozess wieder in den Vordergrund geholt werden. **jobs** zeigt alle im Hintergrund laufenden Jobs der aktuellen shell. (Siehe: Kapitel 17.10, Seite 21 - für das Verwalten aller Jobs, nicht nur derer aus der aktuellen shell).

ctrl-d ist das "Datei-Ende" und wird auch von vielen Programmen bentutzt die Eingabe lesen, um diese Eingabe zu Beenden. Mit dem Befehl **exit** oder mit einem **Ctrl-D** kann eine shell geschlossen werden.

TAB die Tabulator Taste dient zur Vervollständigung von Eingaben. Einmal gedrückt macht sie, falls möglich, eine Vervollständigung: Zweimal gedrückt macht sie Vorschläge. Normalerweise werden Datei- und Verzeichnisnamen und die Namen eingegebener Programme vervollständigt. Inzwischen sind aber auch Bash-Plugins gängig die, z.B. die gängigen Befehlsoptionen von Befehlen kennen und auch Vervollständigen können. Damit ist die Eingabe von Befehlen sehr komfortabel und schnell.

17.4 Wer ist eingeloggt?

who und **w** zeigen dir wer gerade aller am selben Computer eingeloggt ist. **last** zeigt dir wann du zuletzt eingeloggt warst.

17.5 Diverse kleine, nützliche Tools

cal zeigt dir einen aktuellen Kalender. `cal 1 2025` zeigt dir den Kalender von Jänner 2025.

gcal ist ein erweiterter Kalender. `gcal -n -qAT` zeigt dir eine Liste aller österreichischen Feiertage im aktuellen Jahr.

date - zeigt Datum, Uhrzeit und Zeitzone an.

factor - Primfaktoren Zerlegung. Das geht erstaunlicherweise bis hin zu relativ großen Zahlen: `factor 380708481735525870694501390 85771113793`

ping schickt ein ping Packet zum angegeben Server. (Siehe: Kapitel 25.4, Seite 45)

echo gibt die angegeben Argumente wieder aus. z.B.: `echo hallo da`

wc zählt Zeichen, Wörter und Zeilen in einer Datei

seq macht eine Zahlensequenz: z.b: [seq 1 10] die Zahlen 1 bis 10.

md5sum rechnet eine Prüfsumme über den Inhalt der angegebenen Dateien. (praktisch zum sehen ob die gleich sind wie wo anders).

17.6 Ausgabe Umleitung in und von Dateien

Das großer-Zeichen: > leitet die Ausgabe eines Befehls, die ansonsten am Bildschirm landen würde (auch STDOUT genannt) in ein File um.
z.B.:

```
date > heute.txt
```

Würde die Ausgabe vom `date` Befehl in die Datei namens "heute.txt" schreiben. **Achtung:** Falls heute.txt schon existiert wird es kommentarlos überschrieben!

Den Inhalt der neu erstellten Datei können wir uns mit einem Editor oder mit dem `cat` Befehl ansehen: z.B.: `cat heute.txt`

cat gibt ein oder mehrere Dateien am Bildschirm aus. Natürlich lässt sich auch die Ausgabe von cat wieder mit > umleiten. Das Minuszeichen − wird sehr oft an Stellen benutzt an denen ein Filename verwendet werden kann um statt dessen die Eingabe (also die Zeichen die wir gerade eintippen, auch STDIN genann) zu lesen.

`cat - > test.txt` liest die Eingaben die wir tippen und schreibt sie in eine Datei namens text.txt. (Beenden der Eingabe mit **Ctrl-D**).

Analog dazu macht das Kleiner-Zeichen < eine Umleitung einer Eingabe. Ein Programm das normalerweise eine Eingabe lesen würde bekommt nun die Daten aus einer Datei. z.B:

```
seq 1 1000000 > mille.txt
wc < mille.txt
```

Die Zeile mit dem `seq 1 1000000 > mille.txt` erzeugt eine Datei namens mille.txt mit den Zahlen 1 bis 1000000 als Inhalt. (Eine Zahl pro Zeile). Diese Datei leiten wir in das `wc` Tool um zu zählen wie viele Zeilen, Wörter und Zeichen dort drinnen sind.

Verwendet man statt einem > zwei >> so bedeutet das bei der Ausgabeumleitung, dass man eine eventuell vorhandene Datei nicht überschreiben will sondern den Inhalt am Ende **anfügen** will.

```
echo noch eine zeile >> mille.txt
```

macht unsere mille.txt Datei noch um eine Zeile länger.

Fehlermeldungen werden von den meisten Tools nicht über die STDOUT Ausgabe ausgegeben sondern über einen einen Kanal "gesendet". Der wird STDERR genannt und hat im Unix die Nummer 2. Will man dies Ausgabe auch umleiten dann Verwendet man `2 >`. Soll die STDERR Ausgabe wie eine normale Ausgabe behandelt werden so können wir das mit dem Kürzel `2 > &1` machen.
z.B.:

```
ls /etc /gibtsnicht  >bla.txt   2>&1
```

17.7 Wildcards in der Shell

* und ? haben in der shell eine spezielle Bedeutung. Sie sind "Wildcards" in Datei- und Verzeichnisnamen. Ein Stern * wird durch ein oder mehrere Beliebige Zeichen ersetzt. Ein Fragezeichen ? wird durch genau ein beliebiges Zeichen ersetzt:

cat *.txt gibt alle files aus dem aktuellen Verzeichnis, die auf .txt enden aus.

gimp bild?.jpg ruft das Programm **gimp** auf und übergibt alle Dateien die mit bild beginnen und dann genau ein zeichen haben. Also bild1.jpg und bildx.jpg aber **nicht** bild22.jpg.

17.8 Dateien kopieren, verschieben und verlinken

cp steht für copy. `cp quelle.txt ziel.txt` würde die Datei quelle.txt auf die Datei ziel.txt kopieren. (**Achtung:** falls die Datei ziel.txt schon existiert wird sie überschrieben). Ist das letzte Argument ein Verzeichnis so werden die Files (es können in diesem Falle mehrerer sein) mit dem selben Namen ins Zielverzeichnis kopiert. (Wenn mehr als 2 Argumente angegeben werden, so muss das letzte Argument ein Verzeichnis sein). cp hat viele Optionen u.a. die Option -r für "rekursiv", falls Verzeichnisse samt aller Unterordner kopiert werden sollen.

mv funktioniert ähnlich wie cp, aber es wird nicht kopiert sondern verschoben (move).

ln -s erstellt einen, so genannten "Symlink". Das schaut aus wie eine Datei ist aber nur ein Verweis auf eine Datei oder ein Verzeichnis. Das ist praktisch um sich z.b. "Abkürzungen" zu erstellen. z.B.: `ln -s /usr/share/sounds Desktop/klaenge` würde einen Link namens "klaenge" im Verzeichnis Desktop erstellen das auf den Systemordner zeigt in dem Linux viele Audiofiles abgelegt hat.

17.9 Hilfe!

Neben Doktor google, gibt es auch direkt im Linux viel eingebaute Hilfe.

Viele Befehle haben direkt Hilfe eingebaut. Mit den Optionen **-h** –**help** oder **-help** (leider nicht einheitlich) wird eine kurze Hilfe angezeigt. Diese Hilfe ist meist nur kurz und listet meist nur die Optionen auf.

Der **help** Befehl zeigt Hilfe zur aktuellen shell (BASH) an. **help echo** würde z.B. zum (intern in der shell eingebauten) echo Befehl anzeigen. Das funktioniert aber nur bei Befehlen die direkt, intern in der shell eingebaut sind. Für die meisten Befehle brauchen wir daher eine andere Hilfe:

man ist der Befehl zum Aufruf der **man**ual Seiten. Die meisten Befehle haben solche man-pages.

`man echo` würde z.b. die man page zum echo Befehl ausgeben (der neben der eingebauten echo Funktion ebenfalls existiert).

`man -k suchbegriff` sucht in den Man-Pages nach "suchbegriff".

man verwendet zur Anzeige der Seiten einen so genannten "Pager". Da viele man-pages sehr lang sind kann man/frau damit in den Pages blättern.

Hier die wichtigsten Keyboard-Shortcuts für den Standard Pager. Der standard-Pager im Linux ist "less" und ist eine Weiterentwicklung von "more".

Leertaste zum weiterblättern.

q beenden

Pfeiltasten oder j k um jeweils eine Zeile nach unten oder oben zu verschieben.

/suchstring sucht nacht "suchstring". (Wobei "suchstring" eine so genannte "regular expression" ist - mehr dazu später).

n die vorgehende Suche wiederholen (weitersuchen -**n**ext).

N weitersuchen nach oben (also in die andere Richtung).

Auf einer grafischen Oberfläche kann auch das Gnome-Help tool "yelp" verwendet werden: z.b.: `yelp man:ls`

Viele GNU Tools haben neben einer man-Page auch eine "info" Page. **info** ist etwas mühsam zu bedienen, aber kann dafür Links zu anderen info Seiten enthalten. Eine frühe Form von "hypertext".[10]

[10] Der KDE Eigene Webbrowser konqueror erlaubt die Eingabe von `man:` und `info:` URLs im Browser die direkt zu info und man Seiten führen. z.B.: `info:seq`

17.10 Prozessmanagement

Um zu sehen was auf deinem Linux gerade so läuft:

```
ps uax
```

Das **ps** Kommando zeigt dir die aktuell laufenden Prozesse im System. Die Optionen "uax" und "elf" werden oft verwendet um mehr Info zu sehen als ps alleine anzeigt. Das **pstree** Tool zeigt eine Baumstruktur der Prozesse an (wer welchen gestartet hat).

Oft ist interessant zu sehen wer gerade die meiste CPU verbraucht. Dafür ist **top** nützlich. top ist ein interaktives Tool und listet die Prozesse nach ihrer CPU Auslastung. (Alternativ wird nach Speicherbedarf sortiert: nach dem Starten von top **M** (groß) drücken). Ein kleines q (quit) oder ein Ctrl-c beendet das Tool.

Alle Prozesse haben eine Nummer. Diese kann benutzt werden um die Prozesse zu steuern. z.B. um ihnen zu sagen sie sollen sich beenden:

`kill 1234` würde dem Prozess mit der Nummer 1234 sagen, er soll sich bitte beenden. Notfalls muss das Betriebssystem mit dem Holzhammer nachhelfen: `kill -9 1234` hilft dann.

17.11 CLI für Vortgeschrittene

Die bisher gelernten Befehle reichen für einfache Aufgaben. Wer etwas mehr in der Shell machen will benötigt aber noch einige Extras. Hier die wichtigsten davon:

17.12 Mit ssh auf einem anderen Rechner arbeiten

Mit

```
ssh username@hostname
```

kann eine verschlüsselte Konsole Verbindung zu einem anderen Linux Rechner aufgebaut werden. Dort können wir dann so arbeiten wie am lokalen Rechner. Mehr dazu im eigenen SSH Kapitel.

17.13 Dateien suchen mit "find"

Find haben wir schon kurz erwähnt. Ohne Argumente listet es alle Datei und Verzeichnisnamen ab dem aktuellen Verzeichnis auf. Ein Argument wird als Verzeichnis gelesen, ab dem gesucht werden soll. z.B.:

```
find /etc
```

Die Liste ist dabei meist recht lang. Um spezifischer suchen zu können gibt es sehr sehr viele Argumente im find. Hier einige der wichtigsten um die Suche einzuschränken:

find -type f sucht nur Files aber keine Verzeichnisse.

find -type d sucht nur Verzeichnisse (Directories).

find -name bla* sucht nur Files die mit bla beginnen. (der Backslash \wird benötigt damit die shell nicht versucht den * durch eine liste aller Dateien zu ersetzen).

find -mtime -3 listet alle Files die in den letzten 3 Tagen (eigentlich 72 Stunden) verändert wurden. (Wichtig ist das Minus es sagt: bis zu 3 Tagen. Ansonsten würden nur Files gelistet die genau 3 Tage (gerundet) alt sind.

find -size +300M würde nur Files listen die größer als 300MB sind. Das ist nützlich um schnell herauszufinden wer den so viel Platz auf der Festplatte verbraucht.

Neben find gibt es auch noch das **locate** Tool. locate sucht überall - aber nicht live sondern nur in einer nächtlich erstellten Datenbank aller files. Dafür aber extrem schnell. Diese Datenbank wird z.B.: nächtlich erstellt und aufgefrischt.

17.14 In Dateien Suchen mit "grep"

Grep erlaubt das Suchen in Files nach so genannten "regular expressions". Die werden wir nicht im Detail behandeln sondern nur die wichtigsten Fälle auflisten. Zum Üben benötigen wir zuerst einmal eine Datei mit viel Inhalt: z.B. `find /etc > uebung.txt` uebung.txt enthält jetzt eine Liste aller Files unterhalb von /etc. Die können wir jetzt durchsuchen:

```
grep ost uebung.txt
```

Damit durchsuchen wir die Dateie uebung.txt und listen alle Zeilen die die Zeichenfolgen "ost" enthalten. Wir können natürlich auch jede andere Zeichenfolge versuchen. Hier die Wichtigsten Optionen für grep, die die Suche und die Ausgabe der Suche beeinflussen:

-i Case**i**ndependent - also Groß/Kleinschreibung ignorieren.

-v re**v**erse - die Suche umkehren. Es werden alle Zeilen ausgegeben die den text **nicht** enthalten.

-c count - es wird ausgegeben wie oft die Zeichenfolge vorkommt

-n li**n**e number - es wird die Zeilennummer mit ausgegeben

-2 es werden 2 Zeilen vor und nach dem Treffer mitangezeigt.

Die Optionen lassen sich natürlich auch kombinieren. z.B.:

```
grep -3ni ost uebung.txt
```

Zeigt alle Files die die Zeichenfolge "ost" enthalten, ignoriert dabei Groß/Kleinschreibung, zeigt 3 Zeilen vor und nach jedem Treffer und gibt bei der Ausgabe die Zeilennummer mit an.

```
grep -c Arbeit mew*.txt
```

Hier noch einige Beispiele für "Regular Expressions":

blabla Sucht nach genau dem Text "blabla".

bl.bla Würde auch blibla und bl7bla finden. Der Punkt ist Joker für ein beliebiges Zeichen.

bl[0-9]bla findet nur Zeichenketten die an der dritten Stelle eine Ziffer haben.

bla*bla findet blbla blabla blaaaaaaabla. Der * bedeutet: eine beliebige (inkl. 0) Anzahl des vorigen Zeichens.

^bla Nur Zeilen die mit bla **beginnen**

bla$ Nur Zeilen die mit bla **enden**.

17.15 Mit einer "Pipe" - Ausgabe von einem Programm zum nächsten weiterleiten

Wir haben bereits gelernt wie wir die Ausgabe eines Programms in ein File umleiten (>) und wie wir die Eingabe zu einem Programm aus einem File lesen (<). Oft wollen wir uns den Zwischenschritt ersparen und einfach die Ausgabe eines Programms an ein weiteres Programm leiten. Dazu dient der Pipestrich (|). z.B.:

```
find . | grep Dok
```

Hier verwenden wir das vorher gelernte grep Tool um die lange Ausgabe von find etwas zu filtern. Natürlich können zusätzliche Pipes benutzt werden um die Ausgabe wieder Weiter zu leiten. z.B.:

```
find . | grep Dok | grep -v [0-9]$
```

Würde aus der vorigen Ausgabe noch alle Zeilen entfernen (-v) die mindestens mit einer Ziffer enden.

17.16 Nützliche Tools für die Pipe

Die meisten der Tools hier können auch direkt mit einem oder mehreren Files als Argument benutzt werden. Sehr praktisch sind sie aber auch innerhalb einer Pipe:

less ist ein "Pager" (Siehe: Kapitel 17.9, Seite 20) Less ist die Weiterentwicklung des alten Unix Tools "more". Mit dem Pager kann in der Ausgabe geblättert werden.

tr Zeichen ersetzen: z.b: `tr A-Z a-z` alle Großbuchstaben durch Kleinbuchstaben ersetzen.

sed der "stream-editor". Ersetzt Zeichenfolgen. z.B.: `sed -es/Kapitalismus/Kommunismus/g`

sort sortiert die Zeilen alphabetisch.

sort -n sortiert die Zeilen numerisch nach der Zahl am Zeilenanfang.

sort -u unterdrückt gleiche Zeilen (**u**nique)

uniq -c zählt wie oft eine Zeile vorkommt (muss zuerst mit sort gefiltert sein).

uniq -u gibt nur Zeilen aus die nicht doppelt vor kommen (welche die doppelt vorkommen werden, werden im Gegensatz zu `sort -u` gar nicht ausgegeben).

Hier ein Beispiel: Angenommen wir haben eine Liste an Namen die sich für eine Veranstaltung angemeldet haben und wir haben eine Zweite Liste von Namen die sich abgemeldet haben. Wir wollen nun eine alphabetisch geordnete Liste aller Namen die nicht abgemeldet sind.

```
cat anmeld.txt  abmeld.txt abmeld.txt | sort | uniq -u
```

Wie funktioniert das? Das cat gibt die anmeld.txt und abmeld.txt aus. Die abmeld.txt aber doppelt. D.h.: Jeder Name der in der abmeld.txt ist kommt mindestens 2 mal in der Ausgabe vor. (Falls er auch in der anmeld.txt vor kommt dann sogar 3 mal). `sort` sortiert die liste und `uniq -u` gibt nur noch die Zeilen aus die nur einmal vor kommen. Damit sind alle Enthalten die Angemeldet waren oder nicht in der Abmeldung aufscheinen. (Zur Sicherheit sollte zuvor überprüft werden ob in der anmeld.txt keine Doppelten vorkommen: `cat anmeld.txt | sort | uniq -d`)

Ein realistischers Beispiel: Wir haben in einem Verzeichnisbaum 2 Millionen Files und davon eine Kopie in der einige Files fehlen oder hinzu gekommen sind und wir wollen sehen welche das sind.

17.17 Environment Variablen

Neben de Argumenten und Optionen die einem Programm beim Starten übergeben werden gibt es noch eine Zusätzliche Möglichkeit den Programmen etwas Mitzugeben. Die so genannten "Environment Variablen".

Die Variablen werden in der shell verwaltet. Variablen die zum "export" bestimmt sind, werden dabei auch an alle gestarteten Programme weiter gegeben. Alle nicht exportierten sind nur intern für die shell selbst sichtbar. Viele der Variablen werden auch benutzt um bash-Funktionen zu Speichern.

Wichtige Befehle zur Verwaltung dieser Variablen in der Bash sind:

set zeigt eine Liste aller Variablen

export zeigt eine Liste aller exportierten Variablen

BLA=sonstwas setzt den Inhalt der Variable $BLA auf "sonstwas".

echo $BLA zeigt den Inhalt der Variable bla. (genau genommen wird der Inhalt der Variablen auf der shell eingesetzt. echo gibt disen Inhalt dann 1-zu-1 wieder aus.)

export BLI=hammer setzt den Inhalt der Variable BLI auf "hammer" und exportiert diese Variable. Nachfolgend aufgerufen Programme in dieser shell können diese Varialbe sehen.

let wochenstunden=7*24 setzt die Variable Wochenstunden auf 168 (let führt einfache (integer) Berechnungen aus)

echo X${BLA}Y zeigt den Inhalt von $BLA aber zuvor ein X und danach ein Y. Die Klammern dienen daher dazu um die Benennung der Variable eindeutig zu machen.

Manche Variablen sind in der shell schon eingebaut und zeigen bestimmte shell-interne Zustände an. Andere werden per Konvention von Verschiedenen Programmen verwendet. Hier eine Liste der wichtigsten Variablen:

$HOME das Home-Verzeichnis der gerade eingeloggten BenutzerIn.

$? der Status-Code des Zuletzt beendeten Befehls (0 = OK).

$PATH Eine mit Doppelpunkten : getrennte Liste aller Pfade in denen Programme gesucht werden, die ohne Angabe des absoluten Pfadnamens gestartet werden sollen. Wenn wir ein Programm starten und nicht wissen woher es ausgeführt wird (In welchen der vielen Verzeichnisse von $PATH es liegt) können wir dem Befehl ein **which** voranstellen.

$HOSTNAME Der Name des Computers. Der kann auch mit dem **hostname** Befehl abgefragt werden.

$EDITOR der default Editor der von vielen Programmen aufgerufen wird.

LANG legt für viele Programme die Sprache fest. z.B.: **en_US.utf8** oder **de_AT.utf8**. Mittels eintippen von **locale** können wir sehen was wir für Sprache eingestellt haben. Wir können auch nur bestimmte Aspekte der Sprach-Einstellungen über die LC_* Variablen steuern.

17.18 Profile

Die Variablen die wir interaktiv setzten sind nur in der aktuellen shell (und falls sie exportiert wurden, auch in den davon aufgerufenen Programmen) gültig. Wollen wir bestimmte Einstellungen permanent machen, so müssen wir die Variablen immer beim einloggen automatisch setzen. Es gibt folgende Möglichkeiten das zu tun:

.profile Die Datei wird beim einloggen von der Bash - und (falls andere shells verwendet werden i.a. auch von diese) geladen.

.bash_profile Falls diese Datei existiert wird die für die BASH verwendet. Die .profile wird in diesem Falle **nicht** gelesen.

.bashrc Diese Datei wird für **jede** neue BASH shell gelesen. Man/Frau sollte daher sehr vorsichtig sein was dort hinein geschrieben wird, da intern oft sehr viele shells aufgerufen werden.

/etc/profile Diese Datei wirkt wie die .profile aber system-weit für alle User. Man sollte daher damit ebenfalls Vorsichtig sein.

Sehr beliebt in den profil-scripts sind **alias** Definitionen. Damit lassen sich Abkürzungen für Befehle definieren. Mit dem Befehl `alias` ohne Argumente können wir uns die aktuellen alias Definitionen ansehen.

17.19 Command Substitution

Wie wir mit dem Pipe (|) Strich die Ausgabe von einem Befehl in den nächsten umleiten haben wir bereits gelernt. Was aber wenn wir die Ausgabe eines Befehls in die Kommandozeile für einen anderen Befehl einsetzen wollen? Das geht in der Bash mit $(`befehlszeile`). Eine Variante die auch in andern Shells funktioniert sind die Backticks. Die einfachen Anführungszeichen nach hinten: `` `befehlszeile` ``.

Einige Beispiele:

```
echo Wir schreiben das Jahr $(date +%Y). Heute ist $(date +%A).
cp $(find Dokumente/ -name \*.jpg -type f) archiv/
```

Der erste Befehl gibt das Jahr und den Wochentag aus. (Es werden dabei spezielle Optionen des date Befehls benutzt.

Der zweite Befehl sucht alle Dateien (keine Verzeichnisse), die mit auf .jpg enden und kopiert sie in ein Verzeichnis namens archiv. Dabei gibt es einen möglichen Stolperstein: Da hier die Namen der Dateien mit Leerzeichen getrennt sind würde der Befehl fehlschlagen falls auch Dateien mit Leerzeichen im Namen vorhanden sind. Einer der Gründe um Leerzeichen in Dateinamen zu vermeiden. Ein weiterer: Die Länge der Befehlszeile ist beschränkt, aber meist kein Problem (Üblich auf einem modernen Linux 2 Millionen Zeichen).

Es gibt jedoch bessere Methoden um Befehle auf sehr viele Files anzuwenden:

17.20 Den selben Befehl auf viele Files anwenden

Viele Befehle erlauben ohnehin mehre Files als Argument. Falls das nicht geht, oder falls die Liste der Files auf die ein Kommando angewendet werden soll erst durch ein Tool erstellt wird, gibt es mehrere Möglichkeiten.

17.20.1 find mit der -exec option

```
find Meinefiles/ -type f -name \*.pdf -exec pdftotext {} \;
```

Obiges würde den Befehl pdftotext für alle pdf Files aufrufen die unterhalb von Meinefiles/ liegen. Das {} wird durch den aktuellen Filenamen ersetzt.

17.20.2 xargs

Das xargs Tool liest Zeilen von STDIN ein und ruft für jede Zeile den Befehl auf der als erstes Argument angegeben wurde. Dabei kann angegeben werden wie viele Argumente maximal übergeben werden. Per default nimmt aber auch xargs Leerzeichen als Trennzeichen. Um das zu Vermeiden kann das Trennzeichen mit der Option -d explizit angegeben werden. Das Zeilenende kann mit \n angegeben werden.

```
seq 1 20 | xargs -n 6 echo
```

der Output würde dann so aussehen:

```
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20
```

`find` kann mit der option `-print0` die Ausgabe mit Ascii-0 zeichen getrennt ausgeben. Diese können in Filenamen nicht vorkommen. `xargs` kann mit der Option `-0` dieses Format lesen. Eine Leerzeichensichere Version der PDF in Text Konvertierung könnte also so aussehn:

```
find Meinefiles/ -type f -name \*.pdf -print0 | xargs -0 -n 1 pdftotext
```

17.21 Ein einfaches shell Script

Obwohl sich sehr viel in "Einzeilern" erledigen lässt ist es oft übersichtlicher und einfacher für komplexere Aufgaben ein so genanntes "shell script" zu schreiben. Ein script kann in verschiedenen Programmiersprachen geschrieben werden aber natürlich auch direkt mit der bash. Hier ein Beispiel für ein sehr einfaches script:

```
#!/bin/bash
# mein erstes shell script
echo heute ist der $(datum).
echo dein homeverzeichnis ist $HOME
```

Was tut das script? Das Kanalgitter # beginnt in der shell einen Kommentar. D.h. was nach dem # kommt wird ignoriert. Die erste Zeile ist somit aus Sicht der shell auch ein Kommentar: `#!/bin/bash`

Die Kombination `#!` sagt allerdings dem Linux dass dieses script mit diesem Programm (eben der bash shell) ausgeführt werden soll.

Die nächste Zeile ist dann ein echter Kommentar und in den letzten Beiden Zeilen wird echo benutzt um eine Ausgabe am Bildschrim zu machen.

Wie können wir das script jetzt starten? Zuerst müssen wir es in ein File schreiben. z.b. `meinscript.sh` und danach müssen wir dem Linux sagen, dass dies ein ausführbares Script oder Programm ist. Dazu der Befehl:

```
chmod ugo+x meinscript.sh
```

Um das script dann wirklich auszuführen müssen wir den Namen des scripts eintippen. Da üblicherweise das aktuelle Verzeichnis nicht im $PATH Pfad ist müssen wir das explizit angeben. Also z.B.:

```
./meinscript.sh
```

Hier noch ein Beispiel für ein etwas komplexeres script, das Files in thumbnails mit der breite 120 Pixel umwandelt.

```
#!/bin/bash
#
# wir wollen im script zaehlen. als zaehler benutzen wir die variable cnt
# die setzen wir am anfang auf 0
cnt=0
```

```
#

# mit einer for schleife koennen wir ueber all angegeben argumente dieses
# scripts durchlaufen. der shell variable hier k wird dabei nach der reihe
# eines der argumte zugweisen.

for k ; do
  echo ich arbeite jetzt mit $k
  let cnt=cnt+1
  # hier geben wir die variable $k aus und erhoehen unseren cnt zaehler
  convert -resize 120  $k .thumbnail.$k
  # das programm convert gehoert zum imagemagik packet und kann bilder
  # in verschiedene formate konvertieren und skalieren.
  # obiger befehl nimmt das originalfile mit namen $k und
  # skaliert es auf 120 pixel breite und nennt das ausgabefile .thumbnail.$k
  # files die mit einem . beginnen sind hidden files.
done
echo wir haben $cnt files konvertiertet
```

Um das script laufen zu lassen müssen wir es wieder ausführbar machen und danach mit den passenden Argumente aufrufen:

```
chmod ugo+x meinthumbnail.sh
./meinthumbnail.sh *.jpg *.png
```

17.22 User, Gruppen und Zugriffsrechte

Im Unix haben alle User einen Usernamen und eine numerische ID. Zusätzlich ist jede/r User Mitglied ein oder mehrere Gruppen. Um zu sehen welchen Gruppen wir angehören können wir den Befehl **id** verwenden:

Die Ausgabe könnte etwa so aussehen:

```
uid=1000(karli) gid=1000(karli) groups=1000(karli),24(cdrom),29(audio),119(bluetooth),23000(staff)
```

Wir sind also Benutzer "karli" mit ID 1000 und der primären Gruppe des selben Namens und der selben ID (1000). Weiters sind wir Mitglieder der Gruppen cdrom,audio,bluetooth und staff. Damit dürfen wir auf die entsprechende Hardware zugreifen und können auf Files zugreifen die der Gruppe "staff" zugeordnet sind.

Eine spezielle Rolle hat der user root (ID 0). Der/die darf alles. Dazu mehr später mehr.

Jedes File gehört immer eine/r BenutzerIn und einer Gruppe.

Die normalen Zugrifssrechte im Unix sind immer Dreigeteilt: Es gibt Rechte die auf den User selbst beschränkt sind dem/der die Datei gehört. Es gibt Rechte für die Gruppe der die Datei gehört und dann noch: Rechte für alle Anderen ("Other" manchmal auch "World").

Die elementaren Rechte sind **r**: Lesen, **w**: Schreiben und **x**: Ausführen (execute). Der **ls -l** Befehl zeigt uns an wem eine Datei gehört und wie die Zugriffsrechte aussehen. Die Ausgabe ist dabei in drei Dreierblocks `rwx` jeweils für User Group und Other. Davor gibt es noch einen Buchstaben der Zeigt welcher Art das File ist. - zeigt normales File: z.B:

```
-rw-r--r-- 1 karli staff 31 Dec 28 15:35 bla.txt
-rw-r----- 1 karli cdrom 31 Dec  8 15:35 bli.txt
-rwxrwx--- 1 karli staff 31 Dec 12 15:35 blo.sh
drwxr-x--- 5 anna  staff 31 Dec 20 17:10 Dok/
```

Die bla.txt darf nur von karli geschrieben werden (w) aber von allen anderen (Group und Other) geschrieben werden.

Die bli.txt darf neben karli auch von allen Mitgliedern der Gruppe "cdrom" gelesen werden.

Die blo.sh sowohl von karl als auch allen Mitgliedern der "staff" Gruppe gelesen, geschrieben und ausgeführt werden, aber von sonst niemanden.

Der letzte Eintrag ist ein Verzeichnis (zu erkennen am d am Anfang) namens Dok. Das Verzeichnis gehört anna die dort auch lesen und schreiben darf. Schreiben in einem Verzeichnis bedeutet hier: Files anlegen. Alle Mitgliede der staff Gruppe dürfen dort auch lesen (d.h. Files auflisten). Das **x** hat für Verzeichnisse eine spezielle Bedeutung. Ein fehlendes x verbietet jeglichen Zugriff auf alles was unterhalb dieses Verzeichnisses

liegt, insbesondere auch auf Files auf die man/frau sonst Zugriff hätte. Niemand außer anna und die Mitglieder der staff Gruppe können also sehen was unterhalb von Dok liegt und können dort auch keine Files lesen, selbst wenn diese ansonsten von "Other" lesbar wären.

17.22.1 Wie verändern wir jetzt die Zugriffsrechte?

```
chown karli:staff bli.txt
```

Andern BenutzerInnen ein File mit **chown** zu geben darf nur der/die root-UserIn. Die Gruppe darf man/frau auch nur auf Gruppen ändern denen man/frau selbst angehört. Obiger Befehl würde die Datei bli.txt der Gruppe staff zuordnen.

```
chmod go-w bli.txt
chmod ugo+r bli.txt
chmod ugo+rwx kommunismus/
```

Obiger Befehl würde für Group und Others die Schreibrechte entfernen (falls vorhanden) und der zweite Befehl würde User, Group und Others Leserechte geben. Der Letzte Befehl gibt Allen alle Rechte auf das "kommunismus" Verzeichnis.

Wichtig für scripts (siehe oben) ist die e**x**ecute Berechtigung für scripts:

```
chmod ugo+rx meinscript.sh
```

Generell darf man/fau nur Rechte auf Files verändern die einem selbst gehören.

17.23 Extended ACLs

Werden im Unix relativ spärlich benutzt. Damit lassen sich auf Dateisystemen die dies unterstützen auch komplexere Rechte realisieren (z.B.: mehrere User die Rechte auf ein File haben obwohl sie nicht Mitglied einer gemeinsamen Gruppe sind). Mit `getfacl` können diese Rechte angezeigt werden und mit `setfacl` gesetzt werden.

```
-rw--r--r--+ 1 karli cdrom 26 Dec 18 15:35 spezial.txt
```

Das + am Ende der Rechteanzeige deutet darauf hin das für dieses File noch spezielle ACLs gesetzt sind.

17.24 root BenutzerIn

Der User mit dem Namen "root" und der BenutzerID 0 darf im Unix so ziemlich alles. Das entspricht in etwa dem "Administrator" im Windows.

Um root zu werden gibt es folgende Möglichkeiten:

17.24.1 Einloggen als root

Sofern wir das Passwort des root Users kennen, können wir einfach als root einloggen. Manche Systeme verbieten das Einloggen von root unter der grafischen Oberfläche, manche auch das root einloggen per ssh.

Auf der Text-Konsole (Ctrl-Alt-Funktionstasten) sollte aber ein root Login möglich sein.

17.24.2 su -

su erlaubt es in der aktuellen shell den User zu Wechseln. Das Minus bedeutet dabei, dass nach Möglichkeit alles so eingestellt wird wie nach einem normalen Login. Mit **su -** kann man/frau daher root werden, solange das Passwort bekannt ist. **su - andererbenutzer** würde auf einen anderen Benutzer wechseln. Hier muss ebenfalls das Passwort dieses bekannt sein - ausser man/frau ist schon root, dann kann auf jeden Account gewechselt werden.

17.24.3 sudo su -

sudo ist ein Tool, das erlaubt bestimmte Befehle als ein/e anderer BenutzerIn auszuführen. Je nachdem wie sudo konfiguriert ist, darf man/frau dabei alle Befehle ausführen oder nur bestimmte und es wird entweder das **eigene** Passwort abgefragt oder nicht.

Live-CDs und Ubuntu sind i.a. so konfiguriert, dass der/die normale BenutzerIn sudo ohne Passwort verwenden darf. Man/Frau muss sich also kein zusätzliches root Passwort merken und kann dennoch root werden.

Benötigt man/frau nur einen einzelnen Befehle als root so können die direkt mit sudo ausgeführt werden. Wollen wir mehrere Befehle eingeben so ist es am einfachsten sich mittels **sudo su -** gleich eine volle root-shell zu besorgen.

Als root UserIn darf man/frau natürlich auch sudo selbst umkonfigurieren. Das geht mit dem tool `visudo` und ruft den default-Editor auf. Dort kann eingestellt werden, wer sudo aufrufen darf und wer damit welche Befehle aufrufen darf und auch ob ein Passwort für die Verwendung von sudo notwendig ist.

17.25 Passwortwechsel

Siehe: Kapitel 9, Seite 10 (Passwort Sicherheit)

Passwort ändern sollte am besten über die grafische Oberfläche gemacht werden, da manche Distributionen mit dem Login-Passwort auch den Schlüsselsafe schützen. Ansonsten kann das Passwort mit dem **passwd** Tool geändert werden. Dazu muss das alte Passwort bekannt sein. Nur der/die root UserIn darf (alle) Passwörter ändern ohne die alten zu kennen (auch sein/e eigene/s).

18 Mehr nützliche CLI Tools

18.1 Backup, Archive, tar und rsync

tar ist ein Tool zum archivieren von Daten auf Band oder zum "zusammenzippen" von vielen Dateien in einer. In der DOS/Windows Welt ist das .zip-file bekannt. **zip** und **unzip** können auch im Linux verwendet werden, üblicher ist aber die Verwendung von **tar** und die damit generierten Archive mit den Endungen `.tar .tgz .tar.gz .tar.bz2`. Per Konvention nennen wir unkomprimierte Archiv-Files `.tar`, welche die mit dem **gzip** Tool komprimiert wurden `.tar.gz` oder kurz `.tgz` und welche die mit dem bzip2 Tool komprimiert wurden `.tar.bz2`.

Wenn z.b. alle Files unterhalb des Verzeichnisses `Diplomarbeit/` in ein `dipl.tgz` File packten wollen so geht das mit:

```
tar cfvz dipl.tgz Diplomarbeit/
```

Um zu sehen ob das geklappt hat (auflisten der enthaltenen Files in einem vorhandenen .tgz).

```
tar tfvz dipl.tgz
```

Wollen wir das dann z.b. in einem Unterverzeichnis namens "auspacken" auspacken:

```
mkdir auspacken
cd auspacken
tar xfvz ../dipl.tgz
```

Die wichtigsten Optionen von tar (können auch, wie bei anderen Programmen gewohnt, mit - geschrieben werden):

c create - ein neues Archiv anlegen

t test - ein Archiv testen. (Versuchen es Auszupacken ohne es wirklich zu schreiben).

x extract - auspacken

v verbose - während des ein/auspackens die Liste der Files am Bildschirm ausgeben.

z das Archiv sofort mit dem gzip Tool komprimieren.

f file - nicht von einem Magnetband lesen sondern von einem File.

p permissions - nach Möglichkeiten die Zugriffsrechte mitsichern.

Das **gzip** Tool kann auch direkt verwendet werden um eine große Datei zu komprimieren. z.B.

```
gzip langes-file.txt
```

Danach existiert ein File namens `langes-file.txt.gz`. Das kann mit **gzip -d** wieder ausgepackt werden.

Damit das tar-File ein wirkliches Backup ist, muss es natürlich entsprechend sicher Abgelegt werden. z.B.: über ein scp auf einem remote Server.

Um eine Kopie eines Verzeichnisbaumes, der an anderer Stelle aufgehoben wird aktuell zu Halten ist **rsync** ein gutes Tool.

18.2 email

Als grafisches E-Mail Tool verwenden die meisten Linux User den **thunderbird**. Im GNOME gibt es auch noch den **Evolution** - ist sehr Outlookähnlich.

Auf der Textoberfläche ist **mutt** ein extrem guter Email Client.

18.3 chat

Auf der grafischen Oberfläche ist **pidgin** der beliebteste Chat-Client. Auf der Textconsole gibt es **irc**, **irssi** und andere.

18.4 dd - Umgang mit Disk Images

`dd` ist ein Tool das Direkt von Disks lesen und schreiben kann. Damit müssen wir natürlich vorsichtig Umgehen. Sehr schnell kann eine ganze Festplatte gelöscht werden, sofern man/frau als root User arbeitet.

Um z.b. den Inhalt eines USB-Sticks der auf /dev/sdc eingebunden ist, 1:1 in einem File abzulegen würde folgender Befehl dienen:

```
dd if=/dev/sdc of=mein-usb-image.img bs=1M
```

Um ein leeres (mit Nullen gefülltes) File mit der Größe von 20MByte zu erstellen:

```
dd if=/dev/zero of=leer.img bs=1M count=20
```

Die wichtigsten Optionen von dd sind:

if= input File. Von wo die Daten herkommen. Entweder ein Festplattendevice (z.b. /dev/sdc) oder ein Device, dass einen Datenstrom liefert (z.b. /dev/zero liefert lauter Nullen). Es kann aber auch ein reguläres File sein.

of= output File. **Achtung**: Hier kann man/frau ganz schnell einge gesamte Festplatte löschen.

bs= block size. Mit größeren Blöcken geht es normalerweise schneller.

count= wie viele Blöcke. Wird nichts angegeben so wird bis zum Ende gelesen oder geschrieben.

Um z.b. einen bootbaren USB Stick zu machen, kann man/frau sich entsprechende Images von Live-CD Distributionen herunter laden. Diese können dann mit dd direkt auf dem Stick geschrieben werden. Der Stick sollte dabei **nicht** gemountet werden.

Um im Windows ein Disk Image auf einen USB Stick zu Schreiben ist normalerweise ein externes Tool notwendig. z.B: Microsofts Windows USB/DVD Download Tool oder rufus.

19 Drucken

19.1 CUPS - Common UNIX Printing System

Drucken im Linux wird heute üblicherweise mit dem **CUPS** erledigt. CUPS ist ein System (Frei, von Apple entwickelt und auch in OS/X eingesetz) zur Verwaltung von Druckern und von Druckjobs.

Das CUPS System kann mit diversen Tools konfiguriert werden. Am einfachsten ist meist der Direkte Zugriff auf das CUPS Web System: Im Browser auf `http://localhost:631` gehen. Um Drucker hinzufügen zu können muss man entweder als root oder als Bentuzer mit entsprechenden Rechten angemeldet sein. (d.h. es wird im Webinterface User und Passwort abgegragt).

Sinnvoll ist es die erweiterte Version des **footmatic** Packets zu installieren. Das bietet eine erweiterte Auswahl an Druckern und Druck-Filtern.

Hier auch noch einige CLI Tools für CUPS (wie gesagt: es ist auch alles im Webinterface auf `http://localhost:631` sichtbar.

lpstat -s listet die Drucker die vorhanden sind.

lpq listet aktive jobs. Per default am default Drucker. Alternativ kann der Drucker angegeben werden: z.B.: **lpq -Pricoh**

lp filename.pdf Druckt a2ps

lp -d hp17 -o sides=two-sided-long-edge -o media=a4 filename.ps Druckt ein Postscript File namens filename.ps explizit auf A4 papier auf einem Drucker Namens hp17 und Doppeltseitig.

Der Drucker mit dem Namen PDF Druckt in PDF Files die im Homeverzeichnis des Benutzers unterhalb von PDF abgelegt werden. Ist also praktisch um PDFs zu erstellen (Falls ein Programm das nicht direkt kann) und um Testausdrucke zu sparen.

19.2 Postscript Tools

Ursprünglich wurde in Unix vor allem mit Postscript Druckern gedruckt. Es gibt nach wie vor eine sehr große Zahl an Tools die mit Postscript umgehen können. Hier eine kleine Übersicht über einige nützliche PS Werkzeuge:

ps2pdf test.ps Wandelt ein Postscript File test.ps in ein test.pdf um.

evince test.ps Zeigt das Postscript File an.

a2ps -obla.ps test.txt Wandelt den Text in test.txt in ein Postscript File namens bla.ps um. Default ist mit Rahmen und 2-Seitig. Einziger Nachteil. test.txt wird im latin-1 erwartet. a2ps kann leider **keine utf-8** Zeichnsatz. D.H. es muss notfalls vorher umgewandelt werden: `echo test.txt | recode latin1..utf8 | a2ps -o bla.ps`

enscript kann ebenfalls Text in ps Umwandeln, kennt aber viele Encodings.

psnup -2 in.ps out.ps macht auch in.ps ein neues PS namens out.ps indem in.ps 2-Seitig gedruckt wird.

pstops kann seiten auswählen und skalieren. Damit kann z.B aus einem Langen PS ein auf Doppelseitig A4 gedrucktes Booklet erstellt werden. Und viele, viele andere versionen. z.B: **pstops "@1.03(1cm,2cm)" ein.ps aus.ps** Würde die alle Seiten um 3 Prozent vergrößern und alle Seiten um 1cm in X und 2cm in Y Richtung verschieben.

20 Verschlüsselung - Grundlagen

Die Daten in einem Windows oder Linux System sind normalerweise nur durch die Passwortabfrage beim Einloggen geschützt. Wird aber z.b. die Festplatte ausgebaut oder der PC mit einem USB-Stick oder einer Live-CD gebootet, so sind die Daten direkt lesbar.

Ein gutes Passwort ist natürlich wichtig, es schützt aber eben nicht davor wenn Jemand direkten, physischen Zugang zum Computer hat.

Ähnlich ist es mit Daten die Über das Netzwerk übertragen werden. Werden diese im Klartext übertragen so können alle die das Netzwerk abhören können (Internet Provider, der Hersteller deines WLAN Routers, die 13 Jährigen Script-Kiddies im offenen WLAN, Geheimdienste) die Daten lesen.

Wo wird Verschlüsselung Eingesetzt:

https - verschlüsselte Verbindungen zu einem Webserver. Der Webserver auf der anderen Seite kann aber natürlich deine Eingaben lesen. Neben Web werden heute auch viele anderen Protokolle während des Transports verschlüsselt. Auch wenn dein Internet Provider damit nicht mehr direkt deine Eingaben lesen kann: Es ist immer noch nachvollziehbar welche Webseiten du nutzt.

ssh Textkonsole-Verbindungen zu anderen Linux/Unix Computern werden mit dem **ssh** Verschlüsselt. Mehr dazu ein einem Eigenen Kapital: Siehe: Kapitel **??**, Seite **??**.

Festplattenverschlüsselung mit dem LUKS System können in Linux ganze Festplatten verschlüsselt werden. Das ist insbesondere für Laptops nützlich, bei denen es sich oft schwer verhindern lässt das andere Menschen physischen Zugang haben. Der Nachteil hier: Beim Starten muss immer eine Passphrase zur Entschlüsselung eingeben werden.

Datei Verschlüsselung Einzelne Dateien können Verschlüsselt werden. **gpg** - Gnu Privacy Guard ist hier ein nützliches Tool.

Email - Email können mit GPG und S/MIME verschlüsselt werden. Natürlich können z.b. auch einzelne Datei-Attachments (vor dem Versand) händisch verschlüsselt werden.

VPN - Um den Gesamten Internet Verkehr zwischen 2 Standorten zu verschlüsseln wird ein so genanntes VPN (Virtual Private Network) benutzt. Firmen nutzen das z.B. um ihre Laptops von außen ins Firmennetz zu lassen oder um Filalnetzwerke anzubinden. Inzwischen gibt es auch VPN Anbieter die privaten ein VPN bieten. Das geht dann zum Anbieter. Dein Internet Provider und die 13-Jährigen im Starbucks Café können damit nicht mehr sehen wohin du surfst. Dein VPN-Provider natürlich schon.

TOR - Steht für TOR -Onion Routing: Dabei werden alle Internet Daten durch das TOR Anonymisierungsnetzwerk geleitet und mehrfach von einem zum anderen Knoten weitergeleitet. Die Packte kommen bei einem zufällig Ausgewählten Exit-Knoten heraus. Der Webserver den wir benutzten kann damit nicht direkt sehen von welcher IP-Addresse wir kommen. Damit wird obiges Problem (der VPN-Provider sieht wohin du surfst) vermieden. Der Nachteil hier: Manche Internetsits blockieren die Tor-Exit Nodes.

20.1 Wovor Verschlüsselung nicht schützt

Wenn ein/e HackerIn auf deinem Computer eingebrochen ist, dann kann er/sie natürlich alles Lesen was du eintippst und auf alle deine Files zugreifen. Die Festplattenverschlüsselung nützt dir dort gar nichts - weil ja beim laufenden System die Festplatte immer entschlüsselt wird. Deine verschlüsselten Files sind auch nur ein bedingter Schutz: Solange du sie nicht entschlüsselst sind sie sicher. Gibst du aber deine Passphrase für die Entschlüsselung ein ist der Schutz dahin.

Das selbe gilt natürlich für den Schutz der Daten auf der anderen Seite: Wenn wir mit jemanden oder mit einem Server kommunizieren müssen wir auch damit rechnen dass auf der anderen Seite ein/e HackerIn sitzt.

Auch VPN und TOR schützt nicht 100%ig: Mit Bugs in deinem Browser kann unter Umständen ein Webserver herausfinden von welcher echten IP Adresse du kommst.

Auch die Festplattenverschlüsselung schützt nicht 100%ig vor Menschen mit physischen Zugang. Es könnte z.B. jemand in dein Hotelzimmer einbrechen und auf deinem Laptop einen neuen Bootloader installieren, der mit einem Backdoor ausgestattet ist und die Eingegebene Passphrase ausspioniert. (Evil Maid Attack).

20.2 Verschlüsselung als Akt der Solidarität

Auch wenn der Schutz niemals 100%ig ist: Wir sollten doch nach Möglichkeit auf die Sicherheit achten. Wir sollten auch Daten verschlüsseln die nicht besonders oder vielleicht gar nicht geheim sind: Denn wenn nur die Menschen Verschlüsselung benutzen die wirklich heikle Daten haben dann machen sich diese Menschen erst recht Verdächtigt. Insofern macht es Sinn auch die Glückwünsche an die Oma zu Verschlüsseln.

20.3 Verschlüsselungs- und Authentisierungstechniken

Folgende Probleme werden üblicherweise mit cryptografischen Techniken behandelt:

Geheimhaltung - Daten sollen geheim gehalten werden, obwohl sie über unsichere Kanäle übertragen werden.

Integrität - Wir wollen sicher sein, dass Daten nicht verändert wurden.

Authentizität - Wir wollen sicher sein, dass Daten die andere Seite sicher sein kann, dass die Daten von mir Stammen.

Zur Sicherstellung der Integrität werden so genannte Hash-Funktionen benutzt. Die Erzeugen eine Prüfsumme, also einen Digitalen Fingerabdruck von Daten. Idealerweise so, dass es nicht möglich ist den gleichen Fingerabdruck von anderen Daten zu bekommen.

20.4 Hashfunktionen als Prüfsumme

Tools um einen Fingerabdruck von Dateien zu bekommen sind: **md5sum sha1sum sha256sum sha512sum** MD5 ist nicht mehr wirklich sicher aber manchmal ausreichend. SHA512 und SHA256 sind noch sicher.

```
echo test1>bla.txt
echo test2>bli.txt
md5sum bl?.txt
```

```
3e7705498e8be60520841409ebc69bc1  bla.txt
126a8a51b9d1bbd07fddc65819a542c3  bli.txt
```

```
sha256sum bl?.txt
```

```
634b027b1b69e1242d40d53e312b3b4ac7710f55be81f289b549446ef6778bee  bla.txt
7d6fd7774f0d87624da6dcf16d0d3d104c3191e771fbe2f39c86aed4b2bf1a0f  bli.txt
```

Obwohl die Daten sich nur sehr wenig unterscheiden, sind die Hashes total verschieden. Wollen Anna und Berta sicher gehen, dass ihre Daten die selben sind könnten sie sich, z.b. Via Telephon die Hash-Summen durchsagen.

20.5 Symmetrische Verschlüsselung und GPG

In klassischen (symmetrischen) Verschlüsselungstechniken haben die Beiden Parteien die eine Nachricht austauschen wollen einen gemeinsamen, geheimen Schlüssel für die Kommunikation.

Eine, idealerweise etwas längere, Zeichenfolge oder ein Satz wird als so genannte "Passphrase" verwendet. Daraus wird mit einer mathematischen Funktion ein Schlüssel errechnet, der zur Verschlüsselung der Daten verwendet wird.

gpg (GNU Privacy Guard) ist ein Tool mit dem sowohl symmetrische als auch Public-Key Verschlüsselung möglich ist. Um ein File symmetrisch zu verschlüsseln:]

```
  echo streng geheime nachricht > nachricht.txt
  gpg -c nachricht.txt

#  an dieser stelle wird nach einer passphrase gefragt
#  falle es zu einer fehlermeldung kommt:
#
#  export GPG_TTY=$(tty)
#
# damit sagen wir dass GPG am aktuelen terminal $(tty) die
# passphrase abfragen
# darf.
#
# danach existiert eine Datei namens nachricht.txt.gpg
# die können wir z.b. als Email Attachment verschicken.
# oder auf einem USB stick ablegen und weitergeben.
#
# um die daten wieder zu entschluesseln:

  gpg nachricht.txt.gpg

#
# an dieser stelle werden wir wieder nach der passphrase gefragt
# falls wir am selben system arbeiten: gpg kann mit einem
# kleinen programm unsere passphrase cachen
```

20.6 Public-Key Verschlüsselung

Wollen sehr viele Parteien miteinander kommunizieren müsste jeder mit jedem einen Satz geheimer Schlüssel vereinbaren. Das wäre ziemlich hoher Aufwand, vor allem weil die Schlüssel ja zuerst über einen sicheren Kanal (z.b. ein persönliches Treffen) ausgetauscht werden müssten.

Leichter wird das mit so genannten Public-Key Verschlüsselungsystmen. Dort gibt es 2 Schlüsseln: einen zum Verschlüsseln und einen völlig anderen zum Entschlüsseln. Der Schlüssel zum Entschlüsseln kann dabei nicht leicht aus dem zum Verschlüsseln abgleitet werden. Das Schlüssel-Paar (geheimer und öffentlicher Teil) wird i.a. gemeinsam Erzeugt. Damit kann ich z.B.:

Mir einen Schlüssel-Paar machen und den Öffentlichen Teil in der Zeitung abdrucken lassen. Alle LeserInnen können mir dann geheime Nachrichten schicken. Ich kann sicher sein dass nur ich diese lesen kann. Einziges Problem: Die LeserInnen können nicht ganz sicher sein: Die Redaktion könnte eine Eigenes Schlüsselpaar haben und deren Öffentlichen Teil abgedruckt haben. Um sicher zu gehen könnte ich in einem TV interview bestätigen, dass der in der Zeitung abgedruckte Schlüssel auch wirklich meiner ist und ich könnte dort z.B. den sha256 Fingerprint des Schlüssels vorlesen.

Die meisten Public-Key Systeme können auch für digitale Unterschriften verwendet werden. Ich kann also künftig Nachrichten veröffentlichen und die Zeitungsleser können sicher sein, dass die digitale Unterschrift (z.B. ein Verschlüsselter sha256-Fingerprint der Nachricht) wirklich von mir ist.

Das Abdrucken des Schlüssels in der Zeitung ist allerdings nur für Promis praktikablen. Normalerweise werden Schlüssel auf sogennanten Key-Servern veröffentlicht. Dort können sie öffentlich heruntergeladen werden. Dann muss natürlich immer noch überprüft werden ob der Schlüssel tatsächlich der Person gehört, deren Email Adresse im Key angeführt ist.

Dazu gibt es 3 Möglichkeiten:

persönlich den Fingerprint überprüfen das ist am sichersten: Bei einem persönlichen Treffen tauschen wir die Fingerprints des Schlüssels aus. Viele Menschen haben ihren Fingerprint auch in der Email-Signatur und auf ihren Viskitenkarten.

Web Of Trust Man kann seinen Key von anderen unterschreiben lassen. Diese Personen bestätigen, dass der Key von mir ist. Wenn als ein, mir unbekannter Key von Menschen Unterschrieben ist deren Schlüssel von mir Bekannten unterschrieben wurden. Ich verlasse mich hier auf das Web-Of-Trust. Bei so genannten Key-Signing-Parties tauschen Menschen ihre Fingerprints aus und unterschreiben gegenseitig ihre Keys.

Signaturhierarchien ähnlich wie beim Web-Of-Trust vertrauen wir der Unterschrift von Dritten die die Echtheit eines Keys bestätigen. Allerdings nicht in der Art eines Netzes sondern in der Art einer Hierarchie. Einige offizielle Zertifizierungsstellen stellen Zertifikate für die Echtheit aus. So funktionieren z.B. die Zertifikate von Websites oder die von staatlichen Stellen ausgestellten Bürgercards.

20.7 Praktische Public-Key Verschlüsselung mit GPG

Eines der ersten Public-Key Verschlüsselungswerkzeuge war PGP ("Pretty Good Privacy"), das das RSA-System der Allgemeinheit zugänglich machte. Allerdings ist und war PGP nie frei. Als freie alternative gibt es GPG ("GNU Privacy Guard"):

GPG bietet neben symmetrischer Verschlüsselung (Siehe: Kapitel 20.5, Seite 32) alle Funktionen die für Public-Key Systeme benötigt werden:

- Key-Verwaltung und Erzeugung
- Zugriff auf Keyserver
- Verschlüsselung und Signierung von Files

GPG ist ein reines CLI Tool. Es gibt aber grafische Oberflächen dafür. Im Linux z.B. **seahorse** zur Verwaltung und im Gnome-Filebrowser eingebaute (via Plugin: seahorse-nautilus) Verschlüsselung per Mausklick.

GPG startet im Hintergrund ein Programm namens gpg-agent der die Eingegeben Passphrase für den Key merken kann. Um GPG zu erlauben die Passphrase zu lesen ist es eventuell notwendig zuerst

```
export GPG_TTY=$(tty)
```

zu setzen.

Hier die wichtigsten GPG Funktionen:

gpg --full-gen-key Erzeugt ein neues Schlüsselpaar. Es werden verschiedene Optionen abgefragt. Insbesondere ein Name und eine Email Adresse, die dem Schlüssel zugeordnet werden. Ebenfalls eine Passphrase. Die Passphrase wird benötigt um den Schlüssel verwenden zu können. (Der Schlüssel selbst wird in einer Datei unterhalb von .gnupg/ abgelegt, ist aber mit der Passphrase verschlüsselt).

gpg -a -e test.txt Dieser Befehl würde die Datei test.txt Verschlüsseln. Es wird eine Datei `test.txt.asc` erzeugt. Es werden ein oder mehrere Empfänger abgefragt. Die Datei wird für alle Empfänger verschlüsselt. Die dazu gehörigen Public-Keys müssen aber bereits lokal vorhanden sein. (Siehe unten). Die Option **-a** steht für ASCII: Der Verschlüsselte Text ist in einem ASCII lesbaren Format: dies kann z.B.: Direkt in eine Email kopiert werden. Ohne dieser Option würde eine Datei mit der Endung `.gpg` erstellt. Diese kann als Attachment in einer Email versendet werden.

gpg -a -s test.txt ähnlich wie oben aber diesmal wird **nicht** verschlüsselt sondern signiert. D.h. der Text ist nicht geheim aber es kann überprüft werden, dass dieser von mir signiert wurde.

Alternativ kann mit der `--clear-sign` Option gesagt werden, dass der originale Text in der Nachricht im Klartext aufscheint. Oder wir verwenden die Option `--detach-sign` - damit wird eine Signatur erzeugt, die getrennt von der Originaldatei ist. Im Falle einer Email würde man/frau dann sowohl die unverschlüsselte Originaldatei versenden, als auch die Unterschrift dazu.

gpg -es test.txt Macht beides: Verschlüsselung (-e) und Signatur (-s). Es wird eine test.txt.gpg Datei angelegt.

gpgconf --reload gpg-agent der gpg-agent merkt sich die Passphrase. Mit dem reload vergisst er sie wieder.

gpg --list-keys listet alle Schlüssel auf die wir gespeichert (z.b. von irgendwo importiert) haben.

gpg --list-secret-keys listet unsere eigenen Schlüssel auf.

gpg --search-keys name kontaktiert öffentliche Keyserver um nach "name" zu suchen. Mit der Option `--keyserver` wird ein Keyserver angegeben. z.B: `hkp://ipv4.pool.sks-keyservers.net` oder `pgp.mit.edu` Der default Keyserver kann auch in `.gnugpg/gpg.conf` festgelegt werden. (Eine Zeile mit z.B: `keyserver hkp://pool.sks-keyservers.net`).

gpg --recv-keys keyid Hat man/frau mit search einen Key gefunden der passend aussieht oder ist die KeyID bekannt, kann der vom Keyserver heruntergeladen und an den persönlichen Schlüsselbund angehängt werden.

gpg --import somekey.txt Hier würde ein Schlüssel direkt aus einer Datei importiert **ohne** einen Keyserver zu verwenden.

gpg -a --export irgendwer@irgendwo.at damit lässt sich ein bestimmter Key im ASCII Format exportieren. (Die Ausgabe kann mit > in eine Datei umgeleitet werden).

gpg --fingerprint keyid listet den Fingerprint eines Keys.

gpg --list-sigs keyid listet die Unterschriften eines Keys. (Zur Überprüfung des Web-Of-Trusts)

gpg --edit-key Damit können die eigenen Vertrauenseinstellungen bezüglich eines Keys geändert werden. (z.B.: nachdem man/frau den Fingerprint persönlich überprüft hat).

20.8 Steganographie

Um die Tatsache, dass überhaupt verschlüsselte Nachrichten ausgetauscht werden zu verbergen, können diese in anderen Nachrichten versteckt werden. Das nennt sich "Steganographie". Mit dem **steghide** Packet können Texte in Bildern oder Audiodateien versteckt werden.

20.9 Passwortmanager

Es ist ziemlich gefährlich Passwörter mehrfach zu benutzen: Gelingt es Hackern ein Passwort auf einem System zu stehlen ist damit auch die Sicherheit auf den anderen Systemen gefährdet. Wir sollten daher auf jedem System ein anderes Passwort verwenden. Hunderte von Websites wollen, dass wir uns dort mit Username und Passwort anmelden, aber wer kann sich all die Passwörter merken?

Um dieses Problem zu lösen gibt es Passwort Manager.

Dort werden Passwörter in einem Verschlüsselten File gespeichert und bei bedarf können wir dann nachsehen.

Den Passwortmanager selbst sollten wir natürlich nur auf einem sicheren System betreiben. Welche Möglichkeiten gibt es dafür?

pass pass ist ein CLI System das intern mit **gpg** arbeitet und Passwörter in Textfiles speichert, die mit GPG verschlüsselt sind. Einfach praktisch, sicher.

firefox software-security-device Firefox kann sich Passwörter zu Websites merken. Diese werden dann abgespeichert. Um das ganze sicherer zu machen kann ein "Master Passwort" gesetzt werden, mit dem die Passwörter geschützt sind. Hat man/frau viele Passwörter nur im Browser gemerkt so sollte auf ein regelmäßiges Backup der Browserconfiguration geachtet werden.

firefox sync Wer sich bei mozilla einen Account nimmt kann die Passwörter auch online synchronisieren. Passwörter werden dorthin zwar verschlüsselt geschickt. Das Ganze ist zwar bequem aber auch nicht ungefährlich.

google-chrome Kann sich auch Passwörter merken, allerdings **nicht** mit einem Master Passwort - Passwörter können aber im GNOME Keyring (seahorse) abgelegt werden. Chrome synchronisiert diese Passwörter auch online auf anderen Geräten.

seahorse ist der Passwortmanager von Gnome. Keyrings können automatisch, beim Login in Gnome entsperrt werden und von Programmen für automatischen Login benutzt werden. Alternativ kann man/frau dort auch händisch Passwörter managen.

KWallet ist der Keyring/Passwortmanager des KDE Projekts

KeyPassX ist ebenfalls ein freies grafisches Programm und sogar Cross-Platform, d.h.: kann für Linux, Windows und Mac verwendet werden. Auch ein CLI Zugriff ist möglich.

Es gibt als viele Möglichkeiten. Was tun? Ich würde vorschlagen die wichtigsten Passwörter jedenfalls Browser-Unabhängig zu speichern. (z.B. via "pass"). Bei weniger wichtigen (Junk) Websites die ein Passwort benötigen ist es durchaus OK diese auch im Browser zu speichern.

Ganz wichtige Passwörter, das Master Passwort und die GPG-Passphrase sollte man/frau sich aber einfach nur merken. Für entsprechende Online-Accounts sollte auch (Two-Factor Authentication) aktiviert werden (Einloggen mit One-Time-Codes oder Handy-SMS).

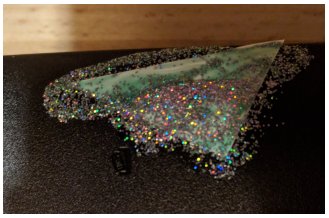
20.10 Festplattenverschlüsselung mit LUKS, Bootpasswort, Bios-Passwort

Gerade auf Laptops sollte die Festplatte mit LUKS (dm-crypt) verschlüsselt werden. Debian bietet dies beim Setup als Option an. Alternativ kann mit dem **cryptsetup** Tool eine Verschlüsselte Partition erzeugt werden.

Beim Starten muss dann immer eine Passphrase zur Entschlüsselung eingegeben werden. Sollen mehre User den selben Laptop benutzen können auch mehrere Passphrases verwendet werden. Um z.B. auf einer vorhandenen Partition einen Key hinzuzufügen:

```
cryptsetup luksAddKey /dev/sdh3
```

Daneben sollte auch ein Passwort für den GRUB-Bootloader gesetzt werden:
`https://www.theurbanpenguin.com/securing-the-boot-process-with-grub-passwords/` und wir sollten im BIOS (dem Menü sofort nach dem Einschalten, in das wir meist durch Drücken des DEL oder F10 Taste kommen.) einstellen dass wir nicht von



USB, CD oder Netzwerk booten wollen und die Einstellungen mit einem Passwort schützen. Paranoide Menschen schützen die Schrauben ihres Laptops noch mit Glitter-Nagellack. (Siehe: Kapitel 20.1, Seite 31 - Evil-Maid Attack).

21 ssh - remote arbeiten mit der secure shell

Wer z.B. einen kleinen Linux Server irgendwo betreibt oder einen Account auf so einem Server irgendwo anders hat oder wer von der Arbeit aus schnell mal einen Zugriff auf die Daten am Computer daheim benötigt, lernt schnell die Vorteiel und Eleganz von **ssh**.

SSH steht für "secure shell". Secure, weil es mit Verschlüsselung arbeitet und "shell" weil es in erster Linie Zugriff auf eine Textoberfläche - shell - gibt.

21.1 Interaktives Einloggen

Angenommen Anna hat einen Account namens "anna" auf dem Server: meinserver.at und kennt ihr Passwort dort.

Dann kann sie, ausser es ist eine Firewall dazwischen, mit dem Befehl

```
ssh anna@meinserver.at
```

und der Eingabe ihres Passwortes dort einloggen und dann auf der Textoberfläche als Benutzerin anna arbeiten.

Angenommen sie ist auf dem Computer von dem aus sie die ssh Verbindung aufbaut auch als User anna eingeloggt, dann kann das `anna@` ausgelassen werden.

Wenn wir jetzt zum testen keinen Account auf einem anderen Server haben können wir natürlich via `ssh andereruser@localhost` auf den eigenen Rechner einloggen und so tun als ob (Eventuell müssen wir dazu erst den ssh-server Teil installieren).

21.2 Wovor schützt das erste s in ssh?

Die Verbindung zum entfernten (remote) Server ist verschlüsselt. Damit ist Anna davor geschützt dass ihr Internet Provider oder die Leute die das selbe WLAN benutzen ihre Verbindung abhören können. Ihr Internet Provider (ISP) sieht allerdings nach wie vor, dass sie eine Verbindung zu diesem Server aufbaut und wie viele Daten dort hin und hinaus gehen. Wir sind ebenfalls davor geschützt, dass unser ISP die Daten zu einem anderen Server umleitet und wir dort versehentlich unser Passwort eingeben, vorausgesetzt wir beachten den Fingerprint. Siehe unten.

Angenommen ihr eigener Rechner wurde gehackt. Dann schützt das ssh natürlich gar nicht, denn die Hacker können all ihre Tastatureingaben und alles mitlesen. Auch wenn sie die Verbindung von einem Internet-Cafe aus aufbaut: Alle Trojans die dort installiert sind können die Verbindung abhören und auch ihr Passwort mitlesen.

Angenommen der Server auf den wir einloggen ist gehackt. Sobald wir mit einem Passwort einloggen kann er Hacker auf der anderen Seite unser Passwort lesen. Das ist nur dann ein zusätzliches Problem falls wir das selbe Passwort auch für andere Server benutze hätten. Aber inzwischen wissen wir ja dass wir das nicht dürfen.

21.3 Dateien kopieren

Mit dem Befehl **scp** können wir über ssh auch Files kopieren. z.B.:

```
scp Pictures/IMG2019*.jpg anna@meinserver.at:/var/www/Bilder
scp anna@meinserver.at:bla.txt .
```

Der erste Befehl würde alle mit IMG2019 beginnenden .jpg Files aus dem Pictures Ordner auf den anderen Server ins Verzeichnis /var/www/Bilder/ kopieren (sofern dieses Verzeichnis existiert und anna dort schreiben darf).

Der Zweite Befehl kopiert die Datei bla.txt aus dem Homeverzeichnis von anna von meinserver.at ins aktuelle Verzeichnis (der Punkt am Ende).

Gnome und KDE Filebrowser haben übrigens auch das ssh Protokoll eingebaut es wir können via Ctrl-L und `sftp://anna@meinserver.at` auch grafisch auf die Daten zugreifen.

21.4 SSH von Mac, Windows, Android, etc

SSH-Clients gibt es auch für andere Betriebssysteme. Hier eine Auswahl:

putty ist ein sehr guter und freier SSH Client für Windows.

ConnectBot ist ein ebenfalls freier SSH Client für Android.

MacOS/X hat ja ein Unix unter der Haube und wird mit dem original ssh Client ausgeliefert.

Terminus, Cathode, Prompt2 sind iOS SSH Apps, allerdings alle Kommerziell.

21.5 Public-Key login mit SSH

Neben dem einloggen mit Passwort kann ssh auch Public-Key Logins.

Dazu generieren wir uns ein Paar aus öffentlichen (.pub) und privaten Schlüssel, danach sagen wir dem Server er soll unseren öffentlichen Schlüssel vertrauen und dann können wir mit unserem privaten Schlüssel einlogen. Klingt recht kompliziert ist aber im ssh sehr einfach:

```
ssh-keygen
ssh-copy-id anna@meinserver.at
```

Das wars. Danach können wir ohne Passwort einloggen. Das **ssh-keygen** Erzeugt unser Schlüsselpaar (in der ssh Terminologie: eine *Identität*). Die Schlüssel werden, wie alle anderen persönlichen ssh Einstellungen unterhalb des `.ssh/` abgelegt. Per default unter dem Namen `.ssh/id_rsa.pub` und `.ssh/id_rsa`. Wir könnten aber, falls wir mehrere Schlüssel wollen, mit der Option `-f` auch einen anderen Namen für den Key angeben - in diesem Falle bekommt der Public-Teil ein .pub am Ende.

Wichtig: Das ssh-keygen fragt nach einer **Passphrase**. Diese wird benutzt um den Key zu verschlüsseln. Hier unbedingt eine gute und lange Passphrase verwenden: gelingt es jemand deinen .ssh/id_rsa private Key zu stehlen und ist dieser nicht mit einer Passphrase geschützt, dann können die HackerInnen auch auf den meinserver.at und alle anderen Server die den Key erlauben.

ssh-copy-id macht ein Login auf den remote Server (hier meinserver.at) und installiert dort den Public-Key (per default: `.ssh/id_rsa.pub`, so dass dieser ohne Passwort einloggen darf. Dazu wird am remote Server im File **.ssh/authorized_keys** eine neue Zeile eingefügt in der der .pub key eingetragen wird. (Das könnten wir ohne dem ssh-copy-id script auch händisch machen.)

21.6 ssh-agent

Moment mal: Wir wollten ohne Passwort einloggen und jetzt müssen wir stattdessen eine noch längere Passphrase verwenden. Wo ist denn da der Vorteil?

Ein Vorteil ist, dass es halbwegs sicher ist den selbe Public-Key auf mehreren Servern zu erlauben. Wir können also mit einem Key und einer Passphrase einloggen. Dass die Passwörter alle unterschiedlich sind braucht uns nicht mehr zu kümmern. Und auch das Eintippen der Passphrase lässt sich reduzieren. Die meisten Distributionen starten beim Login ein Programm namens **ssh-agent**. Der ssh-agent kann die Passphrase "cachen". D.h. wir tippen sie nur einmal ein und können dann für die Dauer unserer Session überall ohne Passwort einloggen.

Mit dem **ssh-add** Tool können wir dem Agent auch sagen dass wir eine Identität (Public/Private Paar) hinzufügen wollen oder mit `ssh-add -D` alle wieder aus dem Cache entfernen.

`ssh-add -l` zeigt uns alle momentan aktiven Identitäten an.

21.7 Der ssh Client ganz persönlich - Die .ssh/config

Im Textfile `.ssh/config` können wir persönliche Einstellungen für SSH ablegen. Besonders nützlich ist es oft sich alias Namen für die verschiedenen Server anzulegen auf denen wir einloggen. z.B.:

```
host web
User=anna
HostName=web-server3.meinserver.at
SendEnv EDITOR LANG LC_*
ForwardX11 no

host bs
User=atsuser7411
HostName=beispielserver123.example.com
RemoteForward 8077 localhost:80
```

Diese Einträge im `.ssh/config` legen Aliasnamen für ssh an. Anna kann ab nun auf den server web-server3.meinserver.at einfach mit `web` ansprechen und den Server beispielserver123.example.com einfach mit `bs`. Also z.B.:

```
scp dok.txt bs:
ssh web
```

Die Einstellung `SendEnv` sagt dem SSH dass es die lokalen Variablen EDITOR LANG und alle Variablen die mit LC beginnen beim einloggen mitübertragen soll. Damit wird auf der anderen Seite der selben default EDITOR und die selbe Sprache verwendet (falls diese dort installiert ist und falls der Server diese Variablen erlaubt).

Die anderen Einstellungen (Forward...) behandeln wir etwas weiter unten.

21.8 Befehle übergeben und Pipes zum Server

Wollen wir auf dem remote Server nur einen einzigen Befehl ausführen, dann können wir den direkt als Argument dem ssh übergeben. z.B.:

```
ssh web "du -hs /var/www/Bilder"
```

Damit loggen wir auf den web-Server ein den wir oben definiert haben und führen nur einen einzigen Befehl aus: `du -hs /var/www/Bilder` (Die Anführungszeichen könnten wir in diesem Falle auch weglassen). Der Befehl würde uns z.B.: den belegten Platz im /var/www/Bilder Verzeichnis anzeigen.

Das erscheint auf den ersten Blick nicht so spektakulär. Viel an Tippaufwand haben wir nicht gespart. Allerdings: Wir können den Pipe Strich | und die Dateiumleitungen < und > Verwenden um Daten zwischen dem lokalen und dem remote System zu übertragen. z.B.:

```
ssh web "cat mein.pdf" | pdftk - cat 17-20 output bla.pdf
```

Mit diesem Befehl verwenden geben wir auf der remote Seite das mein.pdf aus (cat). Die Ausgabe wird über die ssh Verbindug zu uns umgeleitet. Aber da wir ein pdf File am Bildschirm ohnehin nicht lesen können leiten wir es in das pdftk Tool um. Im obigen Beispiel sagen wir dem pdftk Tool es soll seine Eingabe von STDIN (-) beziehen (also die Daten die es per Pipe hineingefüttert bekommt) und dann die Seiten 17 bis 20 Auswählen und auf bla.pdf (lokal) abspeichern. Natürlich geht das auch in die andere Richtung:

```
dd if=/dev/sdh bs=1M  | ssh web "cat - > usbimage.img"
```

Obiges Beispiel würde mittels des dd Programmes vom Gerät /dev/sdh lesen. Das könnte z.B. ein USB Stick sein oder eine alte Festplatte. Da das File recht groß ist wollen wir es nicht lokal abspeichern sondern gleich zum Server übertragen. Wenn nichts anderes Angegeben so gibt dd die Ausgabe gleich auf STDOUT und das wir in diesem Falle mit dem Pipe Strich zur Eingabe von ssh. ssh leitet das über den Verschlüsselten Tunnel auf den remote Server und dort starten wir das Programm "cat -". Das liest die Eingabe (-) und gib es auf STDOUT wieder aus. Die Ausgabe haben wir dann in ein File namens usbimage.img umgeleitet. Wir hätten es aber z.b. Auch dort wieder in ein dd umleiten können um dort z.B.: eine Kopie des USB Sticks zu erzeugen.

21.9 Befehlsverweigerung mit SSH

Das direkte Angeben von Befehlen ist recht nützlich, insbesondere wenn wir Abläufe automatisieren wollen. D.h. Wir wollen z.b. Immer um 3 Uhr nachts ein Backup machen und mit ssh die Daten auf einen anderen Server kopieren. Oder wir haben einen kleinen Raspberry-Pi der das Garagentor aufmachen kann und den wollen wir vom Handy aus aktivieren. Oder wir wollen einer externen Firma erlauben das Bild einer Webcam auf unserem Server upzudaten.

Wenn Sachen automatisch, durch ein Script gesteuert passieren sollen, dann ist dort niemand der eine Passphrase eintippen kann - wir würden also einen Schlüssel verwenden der nicht mit einer Passphrase (einer leeren Passphrase) gesichert ist. Aber das wäre erst recht wieder ein Risiko. Um diese Probleme zu lösen gibt es die Möglichkeit einzuschränken was ein bestimmter SSH Key darf:

Im File `.ssh/authorized_keys` stehen die Public-Keys die ohne Passwort einloggen dürfen. Einer pro Zeile. Vor dem Key, also am Zeilenanfang können wir verschiedenen Einschränkungen festlegen.

```
command="/home/anna/bin/updatewebcam.sh",no-port-forwarding,
   no-X11-forwarding,no-agent-forwarding,no-pty ssh-dss AAAAB4....
```

(Das kommt alles in eine Zeile und ist nur hier, der Lesbarkeit halber, umgebrochen). Die wichtigste Einschränkung ist das command= dahinter steht ein Befehl oder ein Script. Und dieses Script oder dieser Befehl ist das einzige das mit dem Key ausgeführt werden darf.

Das updatewebcam.sh Script könnte z.b. so Aussehen:

```
#!/bin/bash
cat - > /var/www/Bilder/webcam.sh
```

Und der Befehl mit dem von der externen Firma das Bild gesendet wird könnte so aussehen.

```
curl http://internewebcam/liveimage.jpg | ssh -i ~/.ssh/webcamkey anna@meinserver.at dummy
```

Mit dem curl Befehl lädt die Firma das Bild selbst übers Netzwerk. z.B.: von einer internen Webcam. Die Ausgabe des Bildes schickt curl auf STDOUT. mit ssh wird das dann zum Account von anna auf meinserver.at weiter geleiet. Der Befehl der hier angegeben wird (dummy) ist egal. Es wird ohnehin immer `updatewebcam.sh` ausgeführt. Mit der Option **-i** kann eine andere Identität - d.h. ein anders Private-Key File angegeben werden. Hier z.B: eines das extra für den Webcam update angelegt wurde. Bei anna ist dort der zugehörige Private Key im authorized_keys File eingetragen.

21.10 Tunnelbau mit SSH

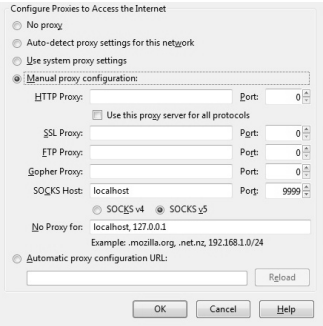
Falls das remote System das erlaubt können mit SSH neben der Shell-Verbindung gleichzeitig auch anderen Verbindungen aufgebaut werden, mit denen andere Daten/Protokolle übertragen werden. Dabei haben wir 4 Arten von Tunnel. Hier jeweils mit Beispiel:

ssh -X web mit der Option X wird die X11 Weiterleitung aktiviert. Danach können am remote Server auch grafische Programme gestartet werden, die dann versuchen ihre Ausgabe über den Tunnel am lokalen X11 Bildschirm anzuzeigen. Wenn wir z.B. von der Firma aus daheim einloggen und dann einen Webbrowser starten wollen mit dem wir die Geräte aus dem Netzwerk Daheim steuern können.

ssh -L 7777:192.168.0.33:80 anna@daheim Angenommen im Netzwerk von anna das über den Server daheim erreichbar ist steht ein Gerät das über eine Website gesteuert werden kann und das die private IP Adresse 192.168.0.33 hat. Nach obigen ist diese Website auch am lokalen Gerät (von dem aus die ssh Verbindung aufgebaut wird) auf Port 7777 erreichbar. Wir könnten dann mit dem Webbrowser auf `http://localhost:7777` surfen und unser Gerät daheim bedienen.

ssh -R 5555:localhost:80 anna@firma Anna hat auf ihrem Laptop daheim eine Website gebastelt die noch nicht online ist. Sie möchte sie aber den KollegInnen in der Firma schon zeigen. Mit der **-R** Option macht sie einen Tunnel dessen Eingang am remote-Server auf Port 5555 Erreichbar ist und der über den Tunnel zu ihr auf den Laptop geht.

ssh -D 9999 anna@firma Anna macht Home-Office, muss dazu aber viele Websites nutzen die nur intern in ihrer Firma erreichbar sind. Mit der Option **-D** baut sie dazu einen so genannten "socks-proxy" auf. In den Einstellungen ihres Webbrowsers kann sie dann den Socks-Proxy verwenden. All ihre Webbrowser-Verbindungen werden jetzt durch den ssh-Tunnel geleietet und scheinen von außen gesehen von ihrem Rechner in der Firma auszugehen.



22 Wie ein Linux Bootet

Linux kann auf sehr viele Arten starten. Welche genutzt wird hängt auch davon ab auf welchem System gearbeitet wird. Am Raspberry-Pi funktioniert Booten ganz anders als auf einem PC. Hier die wichtigsten Stadien des Boot-Vorganges auf einem PC auf dem der übliche Bootloader (grub) verwendet wird:

22.1 Der Bootvorgang

BIOS Am Beginn des Boot-Vorganges steht das, im PC eingebaute BIOS. Das lässt sich i.a. konfigurieren. z.B.: kann man/frau dem BIOS sagen von welcher Festplatte es zuerst Boooten soll und ob es auch versuchen soll von Netzwerk zu Booten oder von USB-Sticks. Beim Boot von Festplatte lädt das Bios den Ersten Sektor der Ersten Platte und startet ihn. Neuere BIOS können so genanntes UEFI-Boot. Das kann zur Not von Linux verwendet werden. Besser ist aber wenn UEFI ausgeschalten wird und man/frau auf "Legacy-Boot" umstellt.

Grub ist der Standard Boot-Loader von Linux. Er sitzt am Beginn der Festplatte und wird von BIOS gestartet. Er bietet ein Menü zur Auswahl der Verschiedenen System an. Seine wichtigste Aufgabe ist es den Linux Kernel und meist auch eine "Initial Ramdisk" (**initrd**) in den Hauptspeicher zu laden und dann anzustarten. Beim Starten des Kernels können auch noch Parameter in der Form `name=wert` übergeben werden, die den weiteren Bootvorgang und den Kernel beeinflussen.

Kernel Der Kernel ist meist komprimiert und er entpackt sich zuerst selbst. Danach wird einiges an Hardware initialisiert. Ist eine initrd RAM-Disk angegeben so wird danach `/initrd /linuxrc` oder `/init` von dieser Ramdisk gestartet. Ist keine initrd angegeben so wird dann der **systemd** gestartet (früher **init**). Mein dem Kernel-Parameter `init=` kann auch ein anderes Programm als erstes gestartet werden. Nützlich ist oft `init=/bin/bash`. Siehe unten.

initiale ramdisk Es gibt sehr viele verschieden Computer und der Kernel muss, um ein Programm starten zu können, zumindest die Festplatte lesen können. D.h. entweder es müssten im Kernel alle Treiber für alle Möglichen Festplattencontroller schon eingbaut sein, oder wir hätten sehr viele Verschiedene Kernel - für jedes Computer-Modell eines. Beides nicht sehr praktisch, insbesondere nicht für Linux-Distributionen die auf möglichst vielen Systemen funktionieren sollen. Zur Lösung dieses Problem dient die initiale RAM-Disk. Das ist ein kleines Linux-System, das bereits vom Boot-Loader zusammen mit dem Kernel in den Speicher geladen wurde. Dort drinnen finden sich i.a. hunderte von Treibermodulen für verschiedenste

Hardware. Das mini-System in der initrd Lädt die passenden Module für die Hardware. Ist das erledigt hat das initrd System seine Schuldigkeit getan und wird vom echten System abgelöst. Es wird das echte Root "/" Filesystem gemountet und mittels pivot_root wird das Mini-System mit dem Echten getaucht. Um all das muss man/frau sich i.a. nicht wirklich kümmern. Das wird in einer guten Distribution alles automatisch angelegt und verwaltet.

systemd Bis vor wenigen Jahren wurde beim Starten eine Linux der "init" Prozess gestartet, der mit einfachen Shell scripts dann den Rest des Systems in Betrieb nahm. Inzwischen wurde der durch ein relativ komplexes System namens **systemd** abgelöst. Der Systemd mountet dann weitere Festplatten, startet weitere Prozesse. z.B. **getty** um auf Text-Konsolen Einloggen zu können, das Netzwerk und dann eventuell auch die grafische Oberfläche.

22.2 Systemd Bascis

Normalerweise muss man hier nicht eingreifen. Aber für Notfälle ein Überblick über die wichtigsten **systemd** Funktionen. Das meiste hier ist natürlich nur für den/die root-UserIn benutzerbar.

systemctl list-unit-files bzw

systemd list-units zeigt einen Überblick über die von systemd verwalteten Services.

systemctl start ssh.service würde z.B. das ssh Service starten (falls es nicht schon gestartet ist). Natürlich gehen auch stop, start, restart und reload).

systemctl enable ssh.service Soll z.B.: das ssh-Service automatisch bei gestartet werden, so muss es mit diesem Befehl "enabled" werden. Natürlich gib es auch ein "disable".

systemctl status ssh.service wenn wir am aktuellen Status und den aktuellen Log Einträge des Services interessiert sind.

journalctl zeigt die aktuellen Log Meldungen die von systemd gesammelt wurden. Wenn bei einem Linux System etwas nicht funktioniert sollten wir zuallererst in die Log Files schauen. `dmesg -T` zeigt uns die Aktuellen Kernel Meldungen (kurz nach dem Booten sehen wir dort auch noch alle Bootmeldungen). `tail -1000 /var/log/messages` oder `tail -1000 /var/log/syslog` zeigt uns die letzten 1000 Zeilen der wichtigsten Log-files.

23 Festplattenpartitionierung und Filesysteme

Beim Aufsetzen einen neuen Linux kann man/frau durchaus die vom Installationssystem vorgeschlagenen default Werte akzeptieren. Insbesondere wenn Linux das Einzige System auf der Festplatte ist.

Es lohnt sich aber doch ein wenig zu Verstehen wie die Aufteilung einer Festplatte erfolgen kann und wie man z.B. Filesysteme von externen USB-Festplatten einbinden kann, ...

Im Windows sind die verschiedenen Festplatten i.a. getrennte Laufwerksbuchstaben (C: D: ...). Im Linux können sie an beliebiger Stelle im Verzeichnisbaum hängen. Um dennoch zu sehen was ein eigener "mount" ist, kann man/frau die Befehle **mount** oder **df** nutzen.

Hier ein Überblick über die Terminologie:

Festplatte Je nachdem welcher Art die Festplatte ist und welcher Treiber sie anspricht haben die Platten verschieden Namen. Üblicherweise ist die erste Festplatte **/dev/sda**.

Partition es gibt eine, von allen Betriebssystemen respektierte Konvention, wie Festplatten aufgeteilt werden. Dazu wird ganz am Beginn Information geschrieben welche Teile die Festplatte hat. Die alte Konvention war ein so genannte MBR-Partitionierung. In dieser Konvention gibt es maximal 4 primäre Partitionen je Festplatte und dahinter noch so genannte Logische oder (extended) Partitionen. Mit dem Befehl `cat /proc/partitions` kann man/frau sich ansehen welche Festplatten und welche Partitionen der Kernel gefunden hat. Falls man/frau dem Kernel sagen will er soll sich das mit den Partitionen nochmal genau ansehen kann der Befehl **partprobe** aufgerufen werden. Zum Ändern einer Partitionierung können die Tools **parted** und **cfdisk** verwendet werden.

Im Bild rechts haben wir 2 Festplatten. Die sda mit 3 Partitionen sda1, sda2 und sda3 und die zweite mit einer Partition sdb1. In neueren Systemen mit großen Festplatten wird auch neuere **GPT** Partitionierungsystem eingesetzt.

UUID Bauen wir in den Computer eine neue Festplatte ein, so ist nicht immer klar welche die neue sda wird. Die Bezeichnung kann sich ändern. Daher ist es oft gut nicht den Namen der Partition anzugeben sondern die UUID - eine interne, eindeutige ID der Partition. Das **blkid** Tool zeigt uns diese IDs.

Filesystem in eine nackte Partition können wir noch keine Files speichern. Dazu muss erst ein so genanntes "Filesystem" eingerichtet werden. (Die Partition muss "formatiert") werden. Linux kann auf sehr viele verschieden Filesysteme zugreifen. "nativ" wird heute meist **ext4** oder **ext3** als Filesystem verwendet. Manche Filesystem sind nur fiktiv (werden vom Kernel vorgegaukelt) und existieren nicht auf einer Festplatte. Diese werden meist als "tmpfs" aufgelistet. Zum einfachen Austausch mit Windows-Systemen wird i.a. **fat32** verwendet.

mount Damit ein Filesystem überhaupt verwendet werden kann muss es ins System eingehängt ("gemountet") werden. Bei USB Sticks und CD-ROMS passiert das meist automatisch (automount). Manchmal will man/frau es aber auch händisch machen. `mount -r /dev/sdc3 /mnt/` würde z.B. versuchen ein auf der Partition sdc3 vorhandenes Filesystem zu mounten und unterhalb von /mnt einbinden. Die Option -r besagt: Read-Only. Da kein Filesystemtype (z.B. ext4) angegeben wurde, versucht Linux den Type selbst herauszufinden.

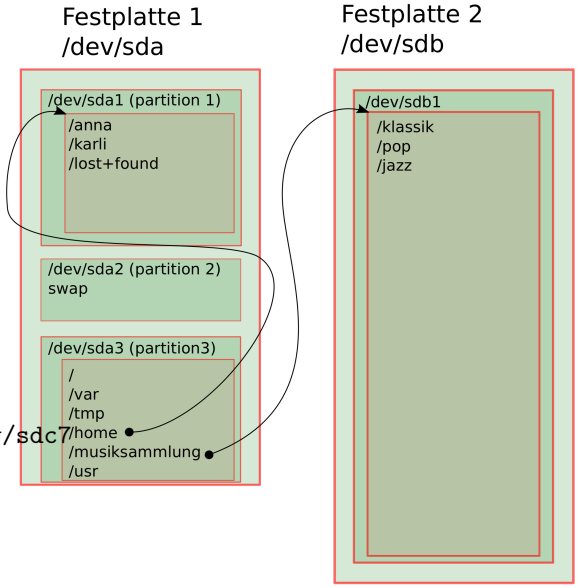
mount point Das Verzeichnis in das ein anderes Filesystem eingehängt wurde heißt "mount-point". Im Bild rechts haben wir 3 Filesysteme. Das Filesystem in sda3 ist das root-Filesystem und ist auf / gemountet. Das Filesystem das auf sda1 vorhanden ist wurde auf /home gemountet und unter /musiksammlung ist das Filesystem in der sdb1 gemountet.

/etc/fstab In diesem File steht drinnen welche Filesysteme wohin gemountet werden sollen.

swap disk Hat der Computer weniger Hauptspeicher (RAM) als die Programme benötigen, kann er Daten auf den so genannten swap Bereich auslagern. Im Linux verwenden wir dafür üblicherweise eine eigene Partition.

23.1 Software RAID

Festplatten haben den Nachteil dass sie leider immer im ungünstigsten Moment sterben. Hat man/frau 2 oder mehr Festplatten eingebaut dann empfiehlt sich die Verwendung eines RAID ("redundant array of independent disks"). Dabei werden die Daten immer auf 2 oder mehr Platten gepiegelt. Das **ersetzt zwar kein Backup** (Daten können immer noch z.B. versehentlich gelöscht werden oder der Computer wird gestohlen, ...) bringt aber dennoch einiges an Datensicherheit. Mit dem **mdadm** Befehl kann aus 2 etwa gleich großen Partitionen ein RAID1 gemacht werden, bei dem die Daten auf beide Festplatten (sinnvollerweise wählt man/frau Partitionen auf unterschiedlichen Platten).




```
mdadm --create /dev/md0 -l1 -n2 /dev/sdb7 /dev/sdc7
```

Würde ein neues RAID Device namens md0 anlegen das aus den Teilen sdb7 und sdc7 besteht.

Wichtig ist es das RAID zu überwachen. Denn der Ausfall einer Festplatte wird sonst leicht übersehen und dann sind die Daten beim Ausfall der zweiten Festplatte doch weg. In /etc/mdadm/mdadm.conf sollten die Konfigurierten RAIDs eingetragen werden. Dort kann auch eine Email-addresse angegeben werden die beim Ausfall verständigt werden soll.

Zusätzlich schadet es nicht öfter mal **cat /proc/mdstat** aufzurufen um den Status des RAIDs zu überprüfen.

23.2 LVM Basics

Die relativ fixe Partitionierung hat ihre Nachteile: Meist wird irgend eine der Partitionen voll und man/frau würde sie gerne Vergrößern aber das geht nicht weil dahinter eine andere Partition liegt.

In einem modernen Setup wird daher schon oft LVM ("Logical Volume Mangement") verwendet. Damit lassen sich Daten sehr flexible über physiche Partitionen verteilen und flexibel erweitern.

Hier die wichtigsten Tools und die Terminologie von LVM

PV - Physical Volume das ist eine Partition die für LVM verwendet werden soll. Damit eine Partition im LVM verwenden werden kann muss sie zuerst als pv markiert werden. z.B.: `pvcreate /dev/sdf7`

VG - Volume Group damit werden 1 oer meherere PV zusammengefasst. `vgcreate meinevg /dev/sdf7 /dev/sdh6` Damite würde eine neue VG erzeugt mit dem namen "meinevg" die aus den PVs sdf7 und sdh8 besteht. Später können neue PVs hinzugefügt, und unter Umständen auch wieder entfernt werden. Da man/frau aber keine direkte Kontrolle darüber hat, wo die Daten zu liegen kommen ist es ratsam die VGs nicht zu bunt über verschiedene Festplatten zu verteilen: Fällt eine davon aus sind u.U. viele LVs davon betroffen. Sollen mehrere PVs zu einer VG verbunden werden, dann sollten diese auf einem RAID liegen.

LV - Logical Volume eine große VG lässt sich in kleinere LVs unterteilen. Dabei sind wir allerdings jetzt recht flexibel. Eine LV kann nachträglich sehr leicht erweitert werden. LVs sollten daher eher kleiner angelegt werden - damit Platz zum vergrößern ist - dort wo er gebraucht wird. `lvcreate -L 20G -n homelv meinevg` - würdein unsere meinvg VG ein neues, 20G großes LV namens "homelv" anlegen. Das Device ist dann wie eine Partition und kann unter /dev/meinevg/homelv oder unter /dev/mapper/meinevg–homelv angesprochen werden.

Neure LVM Versionen unterstützen auch schon direkt innerhalb LVM realisierte RAIDs und es gibt Kontrolle darüber auf welchen PVs bestimmte LVs zu liegen kommen. Snapshots (zu einem bestimmten Zeitpunkt eingefroren Zustände) von LV waren immer schon möglich. Für Traditionelle ("thick") LVs aber sehr langsam. Mit den inzwischen schon stabilen "thin" LVs können auch sehr schnell Snapshots angelegt werden.

Kommt man/frau zu einem unbekannten Linux System und will sich einen Überblick über das dort laufende LVM machen so empfehlen sich die Befehle:

`pvscan`, `vgscan` und `lvscan`. Diese durchsuchen die Festplatten nach LVMs und geben diese aus. Startet man/frau von einer Live-CD und findet auf einem nicht laufenden System ein LVM so müssen die VGs und LVs erst aktiviert werden.

24 Virtuelles Demo-Setup mit KVM

Um die, in den vorigen Kapiteln erlernten Sachen in der Praxis ausprobieren zu können, brauchen wir nicht gleich einen eigens Linux aufsetzen. Wir können das auch in einer simulierten Umgebung tun. Für Virtualisierung gibt es viele Möglichkeiten (Siehe: Kapitel 6.2, Seite 7). Wir machen das hier mit KVM.

24.1 KVM und QEMU

KVM steht für "Kernel Virtual Maschine" und ist eine "hardware beschleunigte" Virtualisierung. D.h: es werden bestimmte Features der CPU benutzt um die Virtualisierung möglichst effizient zu machen. Das virtualisierte System sollte daher nicht all zu viel langsamer Laufen als ein originales.

KVM benutzt sehr viel vom QEMU-System. QEMU ist aber eine reine Software-Virtualisierung. QEMU kann auch andere Computer Systeme (mit anderen CPUs) simulieren. QEMU kann auch auf Mac und Windows Systemen laufen. Ein KVM ist im wesentlichen eine QEMU-Virutalisierung Plus Hardware Beschleunigung.

24.2 Erste Schritte mit KVM

Wenn wir in einem Terminal auf der grafischen Oberfläche einfach nur `kvm` eintippen, starten wir die Emulation eines PCs. Es erscheint ein Fenster in dem ein Boot-Prozess abläuft. Da wir keine virtuelle Festplatte haben klappt das mit dem Boot natürlich nicht.

Klicken wir in das Fenster, dann nimmt sich der Emulator die Maus. Um dort wieder heraus zu kommen können wir **Ctrl-Alt** drücken. **Ctrl-Alt-f** toggelt zwischen Fenster und Fullscreen Modus.

Ein **Ctrl-C** im Textfenster in dem KVM gestartet wurde beendet dies wieder. Alternativ kann KVM auch ganz ohne simulierten Bildschirm gestartet werden.

Auf der Befehlszeile können sehr viele Parameter angegeben werden um verschiedene virtuelle Hardware zu simulieren.

Zuerst wollen wir uns eine virtuelle Festplatte erzeugen:

```
qemu-img create -f qcow2 demo.qcow2 4g
```

Das qemu-img Tool wird hier verwendet um eine neue, simulierte Festplatte im qcow2 Format zu erzeugen die 4G groß ist und im File demo.qcow2 abgelegt ist. Das qcow2 Format erlaubt es Festplatte zu simulieren die viel größer sind als sie eigentlich Platz benötigen und die erst mit dem Schreiben von echten Daten wachsen. (Direkt nach dem Erstellen benötigt unsere 4G große Festplatte nur 193k).

Alternativ könnten wir auch ein Disk Image verwenden in dem der Platz 1:1 dem Entspricht was wirklich auf der virtuellen Festplatte liegt (ein so genanntes **raw** Image). z.B.:

```
dd if=/dev/zero of=demo-raw.img bs=1G count=4
```

Wir wollen uns nun in diesem virtuellen PC ein Debian-Linux installieren und suchen uns nun die `debian-9.6.0-amd64-ne` Installations-CD.

```
kvm -k de -m 2G -cdrom debian-9.6.0-amd64-netinst.iso demo.qcow2
```

Mit diesem Befehl wird jetzt unser kvm wieder gestartet. Wir simulieren eine deutsche Tastatur und 2G Hauptspeicher (hat unser Host Computer weniger als 3G sollten wir hier eventuell weniger verwenden. z.B.: 1G).

Wir simulieren ein CDrom Laufwerk in das unser ISO-Image als CD eingelegt ist und zum Schluss unsere virtuelle Festplatte: demo.qcow2.

Wollen wir 2 Festplatten dann wäre der Syntax z.B.:

```
kvm -k de -m 2G -cdrom inst.iso -drive file=demo.qcow2 -drive file=demo-raw.img,format=raw
```

Hier hätten wir eine qcow2 Festplatte und eine mit einem raw-Image.

24.3 Aufgaben für das Demo Setup

Wir können nun den Debian Setup Prozess mit verschiedenen Optionen ausprobieren. Zum lernen wäre z.b. Folgendes Setup sinnvoll:

- Manuelle statt automatische Partitionierung
- eine 500MB Boot Partition (etwa 10MB Platz vor der Partition)
- Ein Logical-Volume System
- eine rootlv, homelv, tmplv, swaplv
- mit mehreren Disks könnten wir auch ein RAID simulieren
- auch eine Verschlüsselte Partition wäre möglich

Weiters könnten wir

- Log Files ansehen,
- mit apt-get Programme installieren
- ...

24.4 Direkt in eine Shell Booten (init=)

Geben wir am GRUB Boot Bildschirm:

- e drücken
- die Zeile mit den Linux Kernel Parametern editieren und `init=/bin/bash` angeben.
- **ctrl-x** zum booten.
- in der shell haben wir das root Filesystem "read only" gemountet. Um das System schreibbar zu machen `mount -o remount -rw -n /`
- jetzt können wir startup-scripts verändern, oder z.B.: ein neues root-passwort setzen.
- sind wir fertig mounten wir wieder auf read-only: `mount -o remount -r -n /`
- ein **sync** Befehl stellt sicher dass alle Daten auf die Platte geschrieben wurden. danach können wir das System ausschalten (die Virtualisierung abbrechen).

25 Netzwerk Basics

Ein Linux ohne Netzwerkverbindung ist für die Meisten Zwecke nicht wirklich nützlich. Auch wenn das Setup des Netzwerkes in den meisten Fällen automatisch geht ist es praktisch notfalls Probleme selbst diagnostizieren und beheben zu können. Dazu benötigen wir zumindest Grundkenntnisse davon wie Netzwerke funktionieren. Das soll, zusammen mit dem zugehörigen Linux Befehlen hier vermittelt werden.

25.1 Der NetworkManager

Auf Laptops und normalen Arbeitsplätzen wird i.a. der "NetworkManager" gestartet. Der kümmert sich um die Herstellung einer Verbindung. Findet er eine Kabel- (LAN) Verbindung so benutzt er diese. Ansonsten sucht er nach WLAN-Netzwerken und erlaubt mithilfe der grafischen Oberfläche deren Konfiguration. Sind die Netzwerke so eingestellt, dass eine IP-Adresse automatisch zugewiesen (DHCP-Protokoll) wird sollte das ausreichen.

25.2 Das interface

Als Abstraktion von der realen Netzwerk-Hardware (z.B: einer Netzwerkkarte) kennt Linux das so genannte "interface".

Mit dem Befehl `ifconfig` können die vorhandenen Interfaces samt ihrer Einstellung aufgelistet werden. `ifconfig -a` zeigt dabei auch Interfaces die momentan nicht in Verwendung sind, aber vorhanden wären.

Das `ifconfig` Tool ist allerdings nicht mehr ganz taufrisch und man/frau sollte heutzutage eher das **ip** Tool verwenden. Zur Anzeige der Interfaces kann entweder

```
ip -s addr
# oder
ip -s link
```

verwendet werden. `ip addr` listet dabei alle Interfaces samt ihren IP-Adressen. `ip link` zeigt die Hardware (MAC) Adressen. Die Option -s zeigt jeweils auch Statistiken an.

Die IP-Adressen werden zum Datenaustausch ins globale Internet und zwischen lokalen Netzwerken benutzt (OSI-Layer 3). Die MAC-Adessen sind die lokalen Adressen innerhalb des Netzwerkes. Jede Netzwerkkarte hat eine global eindeutige MAC-Adresse. Zur Kommunikation im Internet muss jedenfalls eine IP-Adresse zugewiesen werden.

```
arp -an
# oder
ip neigh
```

Zeigt die Zuordnung von IP-Adressen und MAC Adressen die im lokalen Netzwerk gefunden wurden (also die von anderen Computern im Netz).

Ein typisches kleines Heimnetzwerk sieht z.B.: so aus:

Es gibt einen, meist WLAN-fähigen Router der die Verbindung zum Internet herstellt. Der Router arbeitet intern mit privaten Adressen die nicht im Internet geroutet werden und schreibt diese transparent auf die eine öffentliche Adresse um. Üblicherweise `192.169.0.0/24` Diese Schreibweise gibt zuerst eine Netzwerkadresse an und dann die Zahl der Bits im Netzwerk die zum Netz gehören, die restlichen Adressen dienen dann dazu um Computer im Netz zu unterscheiden. In diesem Beispiel also: `192.169.0.0 bis 192.169.0.255`. Wobei die unterste und die oberste Adresse für spezielle Funktionen reserviert sind.

Netze für den privaten Gebrauch sind:

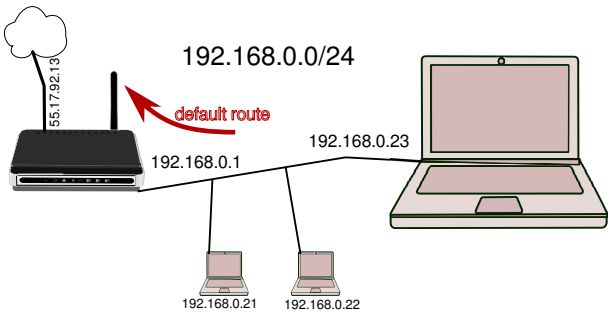
10.0.0.0/8 10.0.0.0 bis 10.255.255.255

172.16.0.0/12 172.16.0.0 bis 172.31.255.255

192.168.0.0/16 192.168.0.0 bis 192.168.255.255

Oder Teile davon. Also z.B.: `192.169.207.0/24` oder `10.119.180.0/24`.

Im gezeichneten Beispiel haben wir das klassische 192.168.0.0/24 Netz. Der WLAN Router hat hier die Adresse .1 (.1 ist für Router üblich aber nicht zwingend). Der WLAN Router vergibt die Adressen für die angeschlossenen Geräte. Hier scheinbar ab .20. Unser Laptop hat hier die .23.



Am Router kann man/frau normalerweise einstellen, dass anhand der MAC Adresse immer die selbe IP Adresse zugeordnet wird - oder es wird eine fixe Adresse am Computer selbst eingestellt - idealerweise nicht aus dem Pool der automatisch vergebenen - um Konflikte zu vermeiden.

Die Adressen der Computer im selben Netz werden auf `ip neigh` auftauchen, sobald sie miteinander sprechen.

Der Computer hat eine so genannte "Routing-Table" die festlegt, wo andere IP Adressen als die lokalen gesucht werden. Für die meisten Heim-Netzwerke haben wir hier nur eine so genannte "default route" die sagt: Alle IP Adressen sind hinter einem bestimmten Router. Im Heimnetzwerk der WLAN Router. Im gezeichneten Beispiel `192.169.0.1`

25.3 Direkte Netzwerkeinstellungen ohne NetworkManager

Debian/Ubuntu etc. haben ihre Netzwerkeinstellunge in `/etc/network/interfaces`

Hie ein Beispiel wie das aussehen könnte:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
     address 192.168.37.79
     netmask 255.255.255.0
     gateway  192.168.37.1

# iface eth0 inet dhcp
```

Im obigen Beispiel wird da so genannte "loopback interface" konfiguriert, das bekommt die Adresse 127.0.0.1 und geht immer zum eigenen Rechner. Ist also für Selbstgespräche gedacht. Danach eine Fix-IP Addresse. Statt der /24 Schreibweise wird die Netzmaske ausgeschrieben. Darunter ist ein gateway angegeben. Der wird dann benutzt um die default route zu setzen.

Es gibt noch viele andere Optionen die nützlich sind. Auch die MAC Adresse kann hier, falls notwendig, verstellt werden. Sehr nützlich ist auch die **up** Option z.B.: `up /root/myfw.sh` . Der Befehl oder das Script nach `up` werden ausgeführt sobald das Interface in Betrieb geht. Dort können wir z.B: Firewall Regeln einfügen.

Auskommentiert ist die Einstellung die die Netzwerkkarte automatisch mit DHCP konfigurieren würde.

Ein wichtiges Element der Netzwerk-Config ist auch der "Name-Server". D.h. Jener Server im Internet der die Umsetzung von Namen (z.B: www.orf.at) auf IP Adressen (z.B: 194.232.104.150) erledigt. Ist dieses Service nicht verfügbar so würden zwar alle IP Adressen funktionieren aber die Internet-Nutzung ist dennoch nicht praktikabel.

`cat /etc/resolv.conf` Zeigt die aktuell eingestellten Nameserver (können auch mehr sein - es braucht aber mindestens einen). Diese Datei wird üblicherweise auch automatisch befüllt (z.B.: via DHCP).

25.4 Fehlersuche im Netzwerk

Hier eine Liste an Befehlen zur Fehlersuche im Netz:

- mit ifconfig oder `ip -s addr` die interfaces auflisten. Hier sehen wir auch ob es eingehende Packte gibt. Vor allem sehen wir auch ob es überhaupt ein Passendes Netzwerk-Interface gibt und ob diesem eine realistisch erscheinende IP Adresse zugeordnet ist.
- wenn es eigentlich ein DHCP im Netzwerk geben sollte dann können wir das auch noch mal händisch mit **dhclient** anwerfen.
- mit `route -n` sehen wir die routen. Wir benötigen jedenfalls eine Default-Route zu unserem Router.
- ein Ping auf den Router oder auf andere Geräte im Netzwerk testet ob dies erreichbar sind: z.B: `ping 192.168.0.1` Das ping zeigt ob die Kommunikation in beide Richtungen zu diesem Gerät möglich ist. Wir würden ebenfalls merken ob Packte verloren gehen.
- ebenfalls nützlich ist ein Ping auf die Nameserver. Dazu sehen wir uns erst mal die /etc/resolv.conf an. Wir können einzelne Nameserver auch gezielt testen mit `host www.google.com IP.des.nameservers` oder mit `nslookup`. Wir können testweise auch mal z.B.: Googles Nameserver eintragen. Der hat eine leicht zu merkende IP `8.8.8.8`. Seit kurzem gibt es auch der Cloudflare Nameserver auf `1.1.1.1`

- manchmal funktioniert unser lokales Netzwerk aber unser ISP hat probleme. Mit dem **tracerroute** Tool können wir den Weg den unsere Packte nehmen verfolgen.
- Wenn das Netz eigentlich funktioniert, wir aber nicht surfen können, dann kann das auch an falschen Proxy-Einstellungen im Web-Browser liegen.

25.5 telnet und netstat

telnet war der Vorgänger von SSH. Es ermöglichte textorientierte Remote-Verbindungen aber ohne Verschlüsselung. Das wird Heute kaum noch verwendet, allerdings ist das Tool weiterhin nützlich um zu testen ob ein Netwerkservice verfügbar ist.

z.B.:

Mittels `telnet www.orf.at 80` können wir uns auf das Port 80 (dort findet sich das unverschlüsselte http Protokoll) verbinden. Wir sehen ob der Server "abhebt" d.h. eine Verbindung überhaupt erlaubt.

Mit `telnet localhost 22` können wir z.B. schnell herausfinden ob auf unserer lokalen Maschine ein SSH (Das Port dafür ist 22) läuft.

Ein Übersicht über alle lokalen, vom Netzwerk aus erreichbaren Services bekommen wir mit:

```
netstat -nt --listen -p
netstat -nu --listen -p
```

Die erste Zeile zeigt TCP-Services (t) und die Zweite UDP-Services (u).

netstat ist auch nützlich um die gerade aktuell aktiven Netzwerkverbindungen anzuzeigen.

25.6 IPv6 Adressen

Im normalen IP (IPv4) Netzwerk gibt es maximal $2^{32} == 4Gig$ Adressen. Bei fast 8 Millionen Menschen hier wird das, trotz NAT nicht reichen. Die neue Version von IP hat dann $2^{128}(ca.3.4 * 10^{38})$ Adressen. Linux unterstützt schon lange IPv6.

Die Adressen sehen in etwa so aus: `fe80::2a6f:cffb:f3a:cef9` Es werden immer 4 Hex Stellen geschrieben, wobei führende 0en weggelassen werden.

25.7 iptables Firewall

Ein Linux mit mehr als einem Netzwerkinterface kann auch als Router arbeiten. Das muss aber zuerst eingestellt werden.

Linux kann auch als Firewall arbeiten: Einerseits um sich selbst zu schützen, falls das Linux als Router arbeitet kann es auch ein Netzwerk schützen. Auch als NAT kann Linux eingesetzt werden.

Es gibt normale 3 Firewall Tabellen: **INPUT** (Um Packte zu filtern die an uns gerichtet sind). **OUTPUT** zum Filtern der lokal versendeten Netzwerkpackte. Die Regeln in der **FORWARD** Tabelle werden nur benutzt wenn das Linux als Router arbeitet und Packte von einem Interface zu einem anderen weiterleitet.

Dann gibt es 2 Tabellen für NAT auf die wir hier nicht näher eingehen.

Hier im schnelldurchgang die Wichtigsten Befehle rund um iptables Firewalls.

```
# zeige die momentan aktiven Regeln
iptables -L

# zeige die aktiven NAT Regeln
iptables -L -t nat

# flush: (alle regeln loeschen)
iptables -F

# zugriff auf das port 22 (ssh) explizit fuer dieses Netzwerk erlauben
iptable -A INPUT -j ACCEPT -s 192.168.0.0/24 -p tcp --dport 22

# zugriff auf das port 22 (ssh) fuer alle anderen netze sperren
iptable -A INPUT -j DROP -s 0/0 -p tcp --dport 22
```

-A fügt neue Regeln am Ende der Tabelle an. (d.h. vorher eingefügte Regeln haben Priorität). ACCEPT erlaubt diese Packte. DROP verwirft sie kommentarlos. -s gibt die Source Adresse an.