Linux-Kurs

Version: 2008-12-29
Autor: Markus Ungermann

Inhaltsverzeichnis

Einleitung 4
Weitere Punkte zu diesem Kurs: 4
Legende oder wie was dargestellt wird 5
Installation von Linux 6
Distribution 6
Installation auf eine einzelne Linux-Platte / Virtuellen PC 7
Partitionen 7
Mountpoint 8
Auswahl der Software Pakete 8
Root und Normalen Benutzer anlegen 8
Bootmanager installieren 8
Der erste Start von Linux 10
Einloggen 10
Textkonsole 10
X-Window 11
Wechsel zwischen Konsole und X Window 11
Verzeichnisse 12
Verzeichnistruktur 12
FHS - Filesystem Hierarchy Standard 13
Umgang mit Dateien 14
Dateien ansehen 14
Verzeichnisse 15
Verzeichnis wechseln 15
Verzeichnisse erzeugen 16
Dateien erzeugen 17
Kopieren 17
Verschieben 17
Löschen 17
Feste und symbolische Links 18
Linux Dokumentation 20
man 20
info 20
The Linux Documentation Project 21
RFC 21
Suchmaschinen 21
Wenn alles nicht hilfst 21
Editoren 22
Textkonsolen Editoren 22
vi 22
GNU nano 22
mcedit 23
emacs 24
X-Window Texteditoren 24
KEdit 24
gnotepad+ 24
XEmacs 25
Mounten 26
Voraussetzungen zum Mounten 26
Mounten 26
Disketten 27
CDROM und DVD 27
umounten 27

Die Datei /etc/fstab....28
Bash....30
Wichtige Tastenkürzel....30
<TAB>-Taste....30
Pfeil hoch/runter Taste....31
"Strg + R"....31
"Strg + L"....31
Umleitungen....31
Pipes....32
Kommandoausführung....33
Joker und Sonderzeichen....33

E - Einleitung

Dieser Kurs wurde von mir für meine Linux-interessierten Arbeitskollegen gehalten. (Dank an meinen Arbeitgeber, die Firma ELZET80).
Ich habe den Kurs geleitet und alle dabei anfallenden Frage weitgehend beantwortet. Diese Doku hier dient als Nachschlagewerk dieses Kurses.
Ich denke aber mit ein bisschen Einsatz kann man diesen Kurs auch alleine am heimischen Rechner gebrauchen.

Hauptziel des Kurses ist es nicht Linux als Anwender zu benutzen, sondern als Administrator einen kleinen Server aufzusetzen und diesen am Laufen zu halten (das ist doch mit administrieren gemeint , oder?). Dafür benötigt Mann/Frau jedoch einigermaßen gute Linux-Kenntnisse und diesen können auch einem normalen Anwender nicht schaden.

Jeder der Fehler findet, Verbesserungen hat, oder mir irgendwas (sinnvolles) sagen will, weitere Themen für den Kurs hat, sollte das auf jeden Fall tun!
Meine Adresse ist: markus@tuxhausen.de.

Weitere Punkte zu diesem Kurs:

- Für die ersten Teile haben wir SuSE Linux 7.0 Professional benutzt
- Wir haben zwei miteinander vernetzte normale PCs benutzt
- Als Literatur-Hilfe für diesen Kurs empfehle ich dieses (wirklich gute) Buch in der neuesten Auflage:

  Michael Kofler
  Linux - Installation, Konfiguration, Anwendung
  Addison-Wesley Verlag
  Online: " Linux" von Michael Kofler

Legende oder wie was dargestellt wird

`[tux]$ ls` `#Kommentar`	Ein normaler Benutzer (hier tux) gibt an der Bash den Befehl **`ls`** ein Alles was hinter # steht ist nur ein Kommentar und darf nicht mit eingegeben werden !
`[tux]/var/log$ ls`	Ein normaler Benutzer (hier tux) gibt an der Bash im Verzeichnis /var/log den Befehl **`ls`** ein
`[root]# ls`	Der Benutzer root gibt an der Bash den Befehl **`ls`** ein
`[root]/var/log# ls`	Der Benutzer root gibt an der Bash im Verzeichnis /var/log den Befehl **`ls`** ein
`man`	Der Name eines Linux-Programms: hier das Programm **`man`**
`command not found`	Ausgabe von Linux oder eines Programms
"Strg+d"	GLEICHZEITIG die Tasten "Strg" und "d" drücken
(Abmelden)	Button abmelden drücken
... bla bla ...	Ausschnitt einer Ausgabe oder einer Datei
#Config-Datei	Inhalt einer Konfigurations-Datei

1 - Installation von Linux

Heutzutage ist eine Linux-Installation kein größeres Problem mehr, wer es schon einmal mit einer aktuellen Distribution gemacht hat wird mir recht geben.
Noch bis ca. 1999 war die Installation wesentlich schwerer, da brauchte man jemanden der es schon mal gemacht, sonst verging dir schnell die Lust.

Distribution

Die Frage ist nur, welche Distribution nimmt man?

Gerade hier hat sich in den letzten Jahren sehr viel getan.
Früher war SuSE im deutschen Raum DIE Distribution schlechthin, dann wurde SuSE von Novell gekauft und konzentriert sich auf Firmen. Aber immerhin bietet Novell das openSUSE an; zum gratis Download oder eben mit Handbuch und 90-Tage Support.

Ungefähr dasselbe trifft auf den Weltmarktführer RedHat zu: Früher hatten die auch eine Version für "normale" Anwender, heute ist alles für Firmen zugeschnitten. Allerdings gibt es von RedHat das Fedora-Projekt , dort unterstützt RedHat die Community und Fedora ist dem RedHat sehr ähnlich, aber eben frei zu bekommen.

Eine weitere große Distribution war die französische Mandrake, ich habe Mandrake immer gerne auf dem Desktop gehabt. Mittlerweile heißt Mandrake Mandriva und kümmert sich auch vornehmlich um Firmenkunden. Aber Mandriva hat auch noch ein Paket mit Handbuch und Support sowie eine One-CD und Free-DVD-Version zum Downloaden.

Einen aktuellen Überblick über Linux-Distributionen gibt der Shop von LinuxLand.

Daneben gibt es tausenden von freien, kostenlosen Distribution, auf einige möchte ich hier kurz eingehen:

Debian
Debian ist schon recht alt und ist besonders gut auf Servern, nicht die aktuellsten Treiber und Software, aber dafür super stabil. Debian setze ich fast immer auf Servern ein. Aber Debian ist nicht das Einsteigerfreundlichste, obwohl sich dort auch viel getan hat.

Knoppix
Knoppix ist ein Linux das komplett auf CD/DVD läuft und keine Installation benötigt. Super als Rettungssystem geeignet, oder um sich einfach mal Linux in Ruhe anzugucken.

Ubuntu
Ubuntu ist zur Zeit (Juni 2007) wohl die angesagteste Linux-Distri. Ubuntu wird von einer Firma entwickelt, aber steht zum freien Download bereit. Auf jeden Fall interessant und einen Versuch wert.

Eine weitere Möglichkeit Linux zu bekommen sind diverse Zeitschriften. Dies hat meistens den Vorteil, das ein bisschen Dokumentation schon dabei ist. Einfach mal im Zeitschriftenhandel gucken.
Empfehlen kann die Linuxe aus der Zeitschrift ct , besonders die Rettungs-CDs und die Knoppicilin-CDs (Knoppix mit Virenscanner) sind klasse.

So, ich erkläre ich mal kurz wie du Linux installierst, dummerweise hängt die Installation ganz von deiner Distribution ab.
Ich gebe hier nur allgemeine Tipps zur Installation, alles weitere ist leider zu Distribution abhängig und zwar von Version zu Version.

Am einfachsten installierst du Linux auf einer eigenen Festplatte, in einem normalen PC (also ein Intel/AMD-Rechner). Wenn diese Platte auch noch die einzige im Rechner ist gibt es kaum Probleme. Aber die meisten können nicht auf Windows verzichten, das stellt aber auch keine größere Hürde (mit Windows 9x/ME) dar:

- Zuerst, Windows9x/ME/2000/XP und Linux laufen gut auf einer Platte (mit `lilo` oder `grub` als Bootmanager), Windows NT ist etwas schwerer
- Zuerst defragmentierst du deine Platte, damit der hintere Teil der Platte wirklich leer ist
- Danach verkleinerst du deine Windows-Partition mit Hilfe deiner Distribution
- Damit ist ein Teil deiner Platte wieder frei und du kannst dort Linux installieren

Was auch ganz ohne Risiko geht, ist einen virtuellen Rechner mittels VMWare oder VirtualPC zu nutzen. Dann hast du sogar zwei Betriebssystem gleichzeitig ;-)

Installation auf eine einzelne Linux-Platte / Virtuellen PC

Wenn dein PC auch noch von CD bootet (was jeder neue tut), geht es ganz einfach los:
Lege die CD in dein Laufwerk und starte den PC neu!
Danach startet die Installation selber und möchte dann irgendwann etwas Platz von deiner Platte.

Partitionen

Da dir die ganze Festplatte zur Verfügung steht, ist das kein Problem, aber du brauchst mindestens **2 Partitionen.** Eine für dein Linux (klar, oder?) und eine als "Festplatten-Zwischen-Speicher" der sogenannte Swap-Speicher. Auf dieser Partition speichert Linux Daten, wenn dein Arbeitsspeicher voll ist . Auch wenn du viel RAM hast, solltest du einen Swap Partition anlegen. Die Größe dieses Swap sollte ungefähr 1-2 fache RAM Größe haben, das sollte normal reichen. Übrigens kannst auf deiner Swap-Partition keine Daten selber ablegen, sondern nur das Linux-System.

Unter Linux heißen die Festplatten nicht c,d,e oder so, sondern werden nach ihrer Anbindung benannt: eine IDE-Platte heißt "/dev/hd*xy*":
x ist die Platte im System: die erste Platte (der Master am Primary IDE Port GLO) heißt "a", die zweite Platte (der Slave am Primary IDE Port) und so weiter.
y ist die Partition auf der Platte, die erste Partition ist 1 und so weiter.
Die ersten 4 Partitionen (1,2,3,4) sind primäre Partitionen, alles andere (5,6,7,8,...) sind logische Partitionen, unter DOS auch logische Laufwerke genannt.

SCSI-Platten heißen fast genau so: sda statt hda; also /dev/sda für die erste Platte

Diese Tabelle sollte weiterhelfen:

/dev/hda	die gesamte erste IDE-Festplatte
/dev/hda1	erste primäre Partition der ersten IDE-Platte
/dev/hda2	zweite primäre Partition der ersten IDE-Platte
/dev/hda5	erste logische Partition der ersten IDE-Platte

/dev/hdb	die gesamte zweite IDE-Festplatte
/dev/hdb3	dritte primäre Partition der zweiten IDE-Platte
/dev/sda	die gesamte erste SCSI-Festplatte
/dev/sdc2	zweite primäre Partition der dritten SCSI-Platte

Lass dich davon nicht abschrecken, du brauchst eigentlich nur 2 Partitionen und ich würde dir folgendes raten:

/dev/hda1 wird deine Hauptlinuxpartition (Root-Partition)
/dev/hda2 wird deine Swappartition

Mountpoint

Wie du eben gelesen hast heißen Platten unter Linux z.B.: /dev/hda1. Die Frage ist nun: Wie komme ich auf diese Platte (oder besser Partition)?
Unter DOS würdest du evtl. `d:` eingeben, und unter Linux `/dev/hda1:` ???
NEIN, so geht das nicht. Unter Linux verwendest du einen Mountpoint.
Ein Mountpoint ist ein besonderes Verzeichnis, schreibst du etwas in dieses Verzeichnis, dann wird es in die entsprechende Partition geschrieben. Auf den ersten Blick mag das komisch klingen, deswegen mal ein Beispiel:
Unser PC hat eine IDE-Festplatte (Master am primären Strang) mit 2 Partitionen, nämlich C und D unter Windows, unter Linux heißen unsere Mountpoints /platte_c (zeigt auf /dev/hda1) und /platte_d (zeigt auf /dev/hda2).
Nun, wollen wir für D (bzw. /platte_d) eine eigene Festplatte einsetzen (nur ein Beispiel ...), bauen diese als Salve am primären Strang ein.
Zuerst Windows: Wenn du Glück hast und Windows nicht wieder alles durcheinander bringt, geht es direkt, wenn nicht kannst du mit den Laufwerksbuchstaben jonglieren ...
Nun Linux: Du änderst den Mountpoint /platte_d, und sagst Linux das dieser nicht mehr /dev/hda2 ist, sondern ab jetzt /dev/hdb1. Das wars, es gibt keinen Ärger da /platte_d ja immer noch da ist.

Damit das allerdings alles klappt, braucht Linux eine "Root-Partition" oder auch "System-Partition" und zwar root im Sinne von Wurzel. Diese Partition ist die Basis aller anderen Verzeichnisse. In der Praxis wirst du schnell herausfinden wie das geht, hier reicht erstmal aus: Du brauchst die Root-Partition und jede Distribution legt sie garantiert mit an.

Auswahl der Software Pakete

Linux selber ist streng genommen nur der Kernel, alles andere sind Programme. Das heißt ohne eine Mindestanzahl von Programmen kannst du mit Linux quasi nichts machen. Darum sprechen viele Freaks auch von GNU/Linux, da ohne die GNU-Software nicht viel unter Linux geht.
Du solltest daher eine Mindestanzahl von Programmen installieren, allerdings macht Linux mit den minimal benötigten Programmen auch keinen Spaß. Du solltest also die Standardprogramme installieren die deine Distribution empfiehlt, damit kannst du schon ganz gut arbeiten

Root und Normalen Benutzer anlegen

Linux ist ein Multiuser-System, d.h. mehrere Leute können gleichzeitig arbeiten (gut zu Hause brauchst du das eher nicht). Diese unterschiedlichen Benutzer haben verschiedene Rechte, der Hauptadministrator „root“ darf auf ALLE Dateien schreibend zugreifen, ein normaler Benutzer aber nicht. Oder anders gesagt, root kann ein System vernichten, ein normaler User (außer ein Cracker) aber nicht. Deshalb solltest du normalerweise nicht als root arbeiten und dir einen extra Benutzer einrichten.

WICHTIG: Vergiss bloß nicht das root -Passwort!!!
Alles weitere über Rechte wird später im Kurs erklären.

Bootmanager installieren

Damit ein Betriebssystem vom PC gestartet wird, muss der PC erstmal wissen was er nach dem Starten tun soll. Dafür gibt es den Bootmanager.
Bei unserer Beispiel-Installation mit einer Platte, kannst du den Bootmanager ruhig auf die Festplatte schreiben, solltest du dagegen Bedenken haben, kannst du ihn zuerst auf Diskette schreiben und später auf die Platte, das ist kein Problem.

So, im Grunde solltest du jetzt bald ein lauffähiges Linux auf deinem Rechner haben...

Hier noch ein netter Link: Leseproben aus dem Buch " Linux" von Michael Kofler, dort gibt es das Kapitel zur Installation als PDF.

2 - Der erste Start von Linux

So, nachdem du dein Linux installiert hast, folgt der erste Start (oder du gehst ins Bett, weil es schon 3h morgens ist ;-)) .
Dein System bootet und zeigt dir jede Menge nützlicher Informationen an, leider viel zu schnell (für alle Ungeduldigen: Es wird mitgeschrieben und zwar in /var/log/dmesg (SuSE: /var/log/boot.msg)).

Einloggen

Nach dem Start wartet eine Anmelde-Aufforderung auf dich, entweder in der Textkonsole oder schöner (da grafisch) ein X-Window-Fenster. Dann wirst du nach deinem Benutzer-Namen und deinem Passwort gefragt.
Dort gibst du jetzt mal deinen Namen (nicht den von root !!) und dein Passwort an, und du bist drin, je nachdem ob die Anmeldung nun auf der Textkonsole oder grafisch war, erwartet dich nun ...

Textkonsole

... der nackte Prompt.
In einer Textkonsole wird normalerweise eine Shell gestartet, davon gibt es unter Linux einige, meistens wird aber die **`bash`** benutzt. Du befindest dich nun in deinem Homeverzeichnis, hier kannst machen was du willst; es ist dein privates Verzeichnis.
OK, gib mal deinen ersten Befehl ein:

```
[tux]$ ls -al
```

Dann sollte etwas in der dieser Art zu sehen sein:

```
...
drwx------15tuxusers4096 Jan 31 10:12 .
drwxr-xr-x5rootroot4096 Jan 31 14:21 ..
-rw-------1tuxusers 99 Jan 31 09:15 .Xauthority
-rw-r--r--1tuxusers5742 Jan 31 09:15 .Xdefaults
-rw-r--r--1tuxusers1305 Jan 31 09:15 .Xmodmap
lrwxrwxrwx1rootroot10 Jan 31 09:15 .Xresources -> .Xdefaults
...
```

Das sind Dateien die sich in deinem Verzeichnis befinden, die meisten haben einen Punkt vor ihrem Namen. Diese Dateien sind versteckt, da sie zur Konfiguration von Programmen dienen und im Normalfall gar nicht sichtbar sein müssen.
Nun gib mal den Befehl **`ls`** ohne Parameter an:

```
[tux]$ ls
```

Wie du siehst, siehst du nichts! Das ist richtig so!
Der Befehl **`ls`** zeigt alle Dateien und Verzeichnisse in dem aktuellen Verzeichnis an, unter DOS heißt er `dir`.
Wird dieser Befehl mit dem Parameter **`-la`** erweitert, zeigt er alle (das "a") Dateien mit weiteren Informationen (das "l").Du kannst diesen Befehl auch so eingeben:

```
[tux]$ ls -a -l
```

Aber **nicht so**:

```
[tux]$ ls al
ls: al: Datei oder Verzeichnis nicht gefunden
```

Klar, die Datei `al` gibt es ja auch nicht!
Mit Minus-Zeichen weiß der Befahl das, das oder die nächsten Zeichen ein oder mehrere Parameter für ihn sind.
Gut, das wars schon für den Anfang.
Wenn du mit deiner Arbeit fertig bist logst du dich aus:

```
[tux]$ logout
```

Oder die Tastenkombination "Strg+d" drücken...
Aber jetzt mach deinen Rechner nicht einfach aus, das mag er nicht (er geht zwar nicht kaputt, aber beim nächsten Mal dauert der Start viel länger , weil er sein Dateisystem reparieren muss)
Wenn deine Distribution die Tastenkombination "Strg+Alt+Entf" richtig gesetzt hat, kannst du damit deinen Rechner herunterfahren, wenn nicht passiert:

- A: Nichts
- B: Dein Rechner macht einen Neustart und steht gleich mit dem Prompt vor dir :-)) (wenn du ihn nicht ausschaltest, während des reboot)

Wenn du A oder B hast, dann musst du dich nochmal einloggen und dann folgendes eingeben:

```
[tux]$ shutdown -h now # oder

[tux]$ halt
```

Geht das auch nicht, ausloggen als Benutzer *root* wieder einloggen und die obigen Befehle (**`shutdown -r now`** oder **`halt`**) eingeben.
Aber dann ist er auch bald aus ;-))

X-Window

... eine schöne bunte Welt mit Fenster, Buttons und meist einem Einführungs-Kurs.
Klicke einfach ein bisschen rum und gehe dann auf ´abmelden´ , das ist unter KDE im K unten links zu finden, oder wenn du die rechte Maustaste auf dem Desktop drückst.
Dann kommt wieder der Anmelde-Bildschirm, willst du den Rechner nun runterfahren (was du aber noch nicht tust) gehe einfach auf herunterfahren.

Wechsel zwischen Konsole und X Window

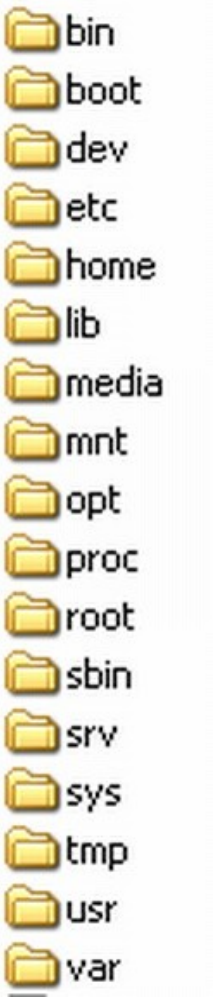
Nun kommen wir zu einem netten Feature eines Multi-User-Systems:
Du hast nicht eine Konsole oder X, du hast beides gleichzeitig und nicht nur eine Konsole sondern (meistens) 6 .
So, nun gehe mal auf die erste Konsole und zwar so:
Bist du in **X Window**, dann drücke "Strg+Alt+F1" und schon erwartet dich wiedermal eine Anmelde-Aufforderung (tja, wer rein will ...),
bist du in einer **anderen Konsole**, dann drücke "Alt+F1".
Spiele mal ein bisschen damit rum und wechsel mal in andere Konsole (F1 -F12), melde dich hier und da mal an und ab.
Dein X-Window findest du übrigens unter F7 wieder.

3 - Verzeichnisse

Verzeichnistruktur

Unixsysteme bestehen aus tausenden von Dateien, damit man überhaupt noch durchblickt werden diese Dateien in Verzeichnisse verteilt. Diese Verzeichnisse beinhalten (meistens) Dateien einer bestimmten Art. Zum Beispiel liegen im Heimatverzeichnisses eines Benutzer meist nur Daten, während die ausführbaren Dateien unter `/bin` liegen.

Das oberste Verzeichnis ist `/`, es ist das Root-Verzeichnis oder Wurzelverzeichnis , alle anderen Verzeichnisse liegen unterhalb des Root-Verzeichnisses:



`/bin`
enthält Programme die jeder Benutzer ausführen darf.

`/boot`
enthält Dateien die zum Booten des Systems benötigt werden. Das sind meist die Dateien des Bootmanagers und des Kernels.

`/dev`
enthält die Device-Dateien. Diese werden benötigt, wenn auf Hardware zugegriffen wird. Denk mal an die Festplatte `/dev/hda1`.

`/etc`
enthält Konfigurationsdateien. Die meisten Linux-Programme kann man mit irgendwelchen Parameter konfigurieren und diese Konfiguration steht in einer Datei; diese Dateien stehen meist in diesem Verzeichnis.

`/home`
enthält die Heimatverzeichnisse der Benutzer. Hier kann der Benutzer seine eigenen Daten ablegen.

`/lib`
enthält die Bibliotheken mit denen die Programme arbeiten.

`/lost+found`
werden beim Festplattencheck Dateien gefunden denen kein Name zugeordnet werden kann, werden diese in diesem Verzeichnis abgelegt. Diese Verzeichnisse gibt es auf jeder Festplatte, allerdings wird es evtl. nach bei Gebrauch angelegt. (Hier habe ich es jedenfalls nicht)

`/media`
alle Medien bzw. Wechseldatenträger sollten unterhalb dieses Verzeichnisses angelegt werden. Früher wurden diese auch gerne nach `/mnt` oder direkt ins Wurzel-Verzeichnis gemountet

`/mnt`
werden Festplatten oder auch Netzwerk temporär/kurzzeitig gemountet, sollten diese unterhalb dieses Verzeichnisses angelegt werden. Aber nicht für temporär Daten !! Diese gehören nach `/tmp` .

`/opt`
meist werden hier kommerzielle Programme oder größere Pakete (Office) angelegt.

/proc
diese Verzeichnis enthält keine echten Dateien !! Es ist quasi als eine Schnittstelle zum Linux-Kernel zu verstehen. In diesem Verzeichnis stehen sehr viele Informationen, einfach mal die Dateien mit einen Betrachter (`less` oder Midnight-Commander).

/root
das Home-Verzeichnis von root. Alle anderen User haben ihre Home-Verzeichnisse unter `/home`.

/sbin
die Befehle in diesem Verzeichnis darf nur root ausführen.

/srv
hier liegen oft, aber nicht immer, die Daten von Serverdienst wie dem Web-Server, FTP-Server, ...
Novell/Suse legen unter `/srv/www` die Daten vom Webserver ab, Debian allerdings unter `/var/www`.

/sys
dieses Verzeichnis gibt es erst seit Kernel 2.6 und ähnelt dem `/proc`-Verzeichnis. Es enthält Kernel-Konfigurationen der PnP-Geräte (PlugAndPlay). Aber es ist weniger interessant als `/proc`.

/tmp
enthält temporäre Dateien, diese dienen zum kurzen Aufbewahren von Informationen.

/usr
quasi alle wichtigen Benutzer-Programme die für den normalen Nutzer wichtig sind liegen, aber auch Quellcodes und Libs. Es ist wohl das größte Verzeichnis, einfach mal reingucken.

/var
enthält Dateien die oft verändert werden. Hier liegen auch die logging-Dateien, oder die Emails.

FHS - Filesystem Hierarchy Standard

Damit nicht jedes Verzeichnissystem total anders unter Unix/Linux aussieht gibt es einen Standard, nämlich den Filesystem Hierarchy Standard.
Aber trotzdem halten sich die Distributionen nur mehr oder weniger daran, aber immerhin ...
Die genaue und aktuelle Spezifikation findest du unter:
http://www.pathname.com/fhs/

Eine gute Erklärung der einzelnen Pfade findet du bei Wikipedia .

4 - Umgang mit Dateien

Dateien ansehen

Einer der meistbenutzten Befehle ist wohl **`ls`** (steht für LiSt), du kennst diesen Befehl schon aus dem vorherigen Teil.
Dieser Befehl hat auch eine Reihe von Parameter, hier mal kurz die Wichtigsten:

Kurzform	Parameter	Beschreibung
`-a`	`--all`	zeigt auch die versteckten Dateien an
	`--color`	zeigt die unterschiedlichen Dateitypen in verschiedenen Farben an (Bei den meisten Distributionen voreingestellt)
`-l`	`--format=long`	zeigt zusätzlich zum Datei-Namen dar, sehr gut zum Kombinieren mit anderen Parametern
`-r`	`--reverse`	dreht die Reihenfolge der Sortierung um, Nützlich mit `-S` oder `-t`
`-S`	`--sort=size`	sortiert nach der Größe , die größte zuerst
`-t`	`--sort=time`	sortiert nach Zeitpunkt der letzten Änderung , die neueste zuerst
`-u`	`--sort=access`	sortiert nach Zeitpunkt des letzten lesenden Zugriffs, muss mit `-t` kombiniert werden
`-X`	`--sort=extension`	sortiert nach Erweiterung (also. txt .html .jpg)

Folgendes solltest du aber beachten, **es gilt für (fast) alle Linux-Befehle**:

- Groß/Kleinschreibung wird unterschieden!
  `ls -r` ist was völlig anderes als **`ls -R`**! (Probiers mal aus !)
- Die meisten Parameter gibt es in einer Kurzform (`-r`) und einer ausführlicheren (`--reverse`), sie sind aber identisch
- Der Parameter `-h` oder `--help` gibt eine kurze Hilfe zum Befehl aus. (Probiers mal mit **`ls`** aus.)
- Der Parameter `-V` oder `--version` gibt eine die Version das Befehl aus. (Ausprobieren!)

Zum Ausprobieren der Befehle, wechselst du am besten ins Verzeichnis `/etc`:

```
[tux]$ cd /etc
```

Die Ausgabe des Befehls **`ls -l gshadow`** (im Verzeichnis `/etc`) werde ich etwas erklären:

```
[tux] /etc$ ls -l gshadow
-rw-r----- 1 root shadow 1279 Jan 31 14:21 gshadow
```

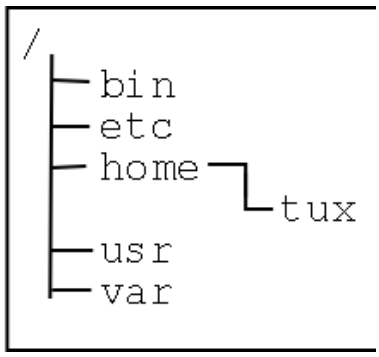
Das sagt uns folgendes:

- Zuerst kommen wird der Dateityp angezeigt: ein "-" steht für eine Datei, ein "d" für ein Verzeichnis (Directory), ein "l" für einen Link
- Danach die Zugriffsrechte für diese Datei: rw-r-----
- Die Anzahl der Hardlinks: 1
- Der Name des Datei-Besitzers: root
- Der Name der Gruppe für diese Datei: shadow
- Dateigröße in Byte: 1279
- Der Zeitpunkt der letzten Änderung: Jan 31 14:21
- Der Name der Datei: gshadow

Was das alles im Klartext heißt, wird im Laufe des Kurses erklärt, fürs Erste sollte es reichen ;-)

Verzeichnisse

Verzeichnisse werden wie Ordner (darum heißen sie auch unter Windows so) genutzt. In einem Verzeichnis kannst du mehrere Dateien oder Verzeichnisse aufbewahren.
Das oberste Verzeichnisse ist das root-Verzeichnis, aber nicht wie der Benutzer root, sondern wie root=Wurzel. All anderen Verzeichnisse hängen unterhalb des root-Verzeichnisses:



Der Befehl **`pwd`** (Print Working Directory), zeigt dir an in welchem Verzeichnis du gerade bist. Außerdem kannst du es an deinem Prompt sehen.

```
[tux] /etc$ pwd
/etc
```

Verzeichnis wechseln

Ich glaube diesen Befehl kennt jeder der jemals an einem Rechner gesessen hat: **cd**, er steht für ChangeDirectory Einfach mal eingeben:

```
[tux] /etc$ cd
[tux] ~$
```

Was ist passiert?
Du bist in deinem Homeverzeichnis gelandet! Das erkennst du daran, dass in deinen Prompt ein "~" steht. Bist du nicht in deinem Homeverzeichnis, steht dort der Verzeichnisname, je nach Einstellung komplett (/usr/local/games/) oder nur der letzte Teil (games/).

Wird **cd** mit einem führenden Slash (/) geschrieben, so ist damit der Pfad unmittelbar unterhalb des root-Verzeichnisses gemeint:

```
[tux] /woauchimmer$ cd

[tux] ~$ mkdir etc   #Verzeichnis bin im Homeverzeichnis erzeugen

[tux] ~$ cd etc      #Gehe ins Verzeichnis etc

[tux] ~/etc$ ls -l
insgesamt 0
```

Wie du siehst bist du im Verzeichnis `etc` unterhalb deines Homeverzeichnisses (also /home/tux/etc) und nicht in `/etc` in dem du eben warst. Dort kommst du so hin:

```
[tux] ~/etc$ cd /etc #Gehe ins Verzeichnis /etc

[tux] /etc$ ls -l
...
-rw-r--r-- 1 root root  2202 Jun 20 2000  DIR_COLORS
-rw-r--r-- 1 root root     6 Jan 25 16:07 HOSTNAME
-rw-r--r-- 1 root root 77540 Aug 2  2000  Muttrc
-rw-r--r-- 1 root root    36 Jul 31 2000  SuSE-release
drwxr-xr-x 2 root root  4096 Jan 31 09:26 SuSEconfig
drwxr-xr-x 2 root root  4096 Jan 25 16:59 WindowMaker
...
```

Desweiteren gibt es noch einen Spezial-Fall: Das Verzeichnis ".." (also zwei Punkte) stellt immer das höher geordnete Verzeichnis da.
Auch hier ein Beispiel:

```
[tux] /etc$ ls -l ../ #zeigt den Inhalt des root-Verzeichnisses (liegt eine Ebene
über /etc) an ...

dwxr-xr-x 2 root root 4096 Nov 20 22:54 bin/
dwxr-xr-x 62 root root 4096 Feb 1 21:46 etc/
drwxr-xr-x 3 root root 4096 Nov 20 22:35 home/
drwxr-xr- 17 root root 4096 Jun 22 2000 usr/
drwxr-xr-x 22 root root 4096 Nov 20 22:16 var/ ..
```

Spiele mal ein bisschen damit herum, denn das musst du draufhaben!

So, ab jetzt werde ich mich etwas kürzer fassen, sonst tippe ich mir noch meine Finger stumpf ;-)). Sollte es deswegen Probleme geben, sofort bei mir melden.

Verzeichnisse erzeugen

Diesen Befehl hatten wir eben schon: **mkdir** und steht für MaKeDIRectory.
Erzeuge mal bitte ein Paar Verzeichnisse in deinem Homeverzeichnis:

```
[tux] ~$ mkdir dir1
[tux] ~$ mkdir dir2
[tux] ~$ mkdir dir3
```

Dateien erzeugen

Hierfür werden wir mal einen Befehl "missbrauchen", **touch**. Normalerweise dient **touch** dazu, den Zeitpunkt des letzten Zugriffes zu verändern. Existiert aber die Datei nicht, wird eine 0 Byte große Datei angelegt und genau das wollen wir!
Dann legt mir mal ein Paar Dateien an:

```
[tux] ~$ touch datei1
[tux] ~$ touch datei2
[tux] ~$ touch datei3
[tux] ~$ touch datei4
```

Kopieren

Dazu dient der Befehl **cp** wie CoPy.
Eigentlich ganz einfach:

```
[tux] ~$ cp datei1 datei11 #Kopiere datei1 nach datei11
```

Willst du ein Verzeichnis, inklusive aller Dateien, kopieren, musst du den Parameter `-r` mitangeben:

```
[tux] ~$ cp -r dir1 dir11 #Kopiere Verzeichnis dir1 nach dir11
```

Verschieben

Dazu dient der Befehl **mv** wie MoVe.
Verschieben ist quasi dasselbe wie kopieren (nur wird die Quelldatei gelöscht) , also ist die Syntax gleich:

```
[tux] ~$ mv datei11 datei12 #Verschiebe datei11 nach datei12
```

Im Gegensatz zu **cp** brauchst du zum Verschieben von Verzeichnissen keinen Parameter anzugeben!

```
[tux] ~$ mv dir11 dir12     #Verschiebe Verzeichnis dir11 nach dir12
```

Löschen

Dazu dient der Befehl **rm** wie ReMove.
Das dieser Befehl gefährlich ist, brauche ich nicht zu erwähnen oder? In der Konsole gibt es keinen Papierkorb...
Löschen einer Datei:

```
[tux] ~$ rm datei12 #datei12 wird gelöscht
```

Willst du ein Verzeichnis, inklusive aller Dateien, löschen, musst du den Parameter `-r` mitangeben:

```
[tux] ~$ rm -r dir11  #dir11 wird gelöscht
```

Vorsicht !!!!
Dieser Befehl ist so mit der gefährlichste den es gibt !!
Verwendest du diesen als root falsch im root-Verzeichnis (/), dann war es das mit deinem System ...

Einen sehr netten Parameter gibt es noch: `-i` bzw. `--interactive` ; jetzt fragt **rm** vor jeder Datei, ob er denn auch wirklich löschen soll:

```
[tux] ~$ rm -i datei1 #datei1 wird mit Abfrage gelöscht, die Antwort ist yes "y"
oder no "n"
```

Feste und symbolische Links

Mit Links kann man auf Dateien verweisen, es gibt die festen und die symbolischen Links.
Bei einem festen Link wird der I-Node gespeichert und bei symbolischen Links wird der Dateiname (evtl. mit Pfad) gespeichert. Feste Links sind schneller und verbrauchen weniger Platz, können aber nur auf demselben Dateisystem angewandt werden, dies ermöglichen die symbolischen Links, die aber langsamer, aber einfacher. Hier mal ein Beispiel:

```
[tux] ~$ ln datei1 datei2 #datei2 ist ein fester Link auf datei1


[tux] ~$ ls -li #-i zeigt den Ort "Ort auf der Platte" an
...
704516 -rw-r--r-- 2 tux user 0 Feb 8 13:12 datei1
704516 -rw-r--r-- 2 tux user 0 Feb 8 13:12 datei2
...
```

Beide Dateien sind eigentlich die selbe, änderst du eine Datei, ändert sich die andere auch, löscht du aber eine Datei, bleibt die andere bestehen !

Etwas anders sieht es bei den symbolischen Links aus:

```
[tux] ~$ ln -s datei1 datei3 #datei3 ist ein symbolischer Link
auf datei1

[tux] ~$ ls -li
...
704516 -rw-r--r-- 2 tux user 0 Feb 8 13:12 datei1
704891 -rw-r--r-- 2 tux user 0 Feb 8 13:12 datei3 -> datei1
...
```

Der Unterschied ist recht deutlich zu sehen, es wird auf eine Datei gezeigt.
Im Normalfall solltest du symbolische Links verwenden, da sie einfacher zu handhaben sind.

Gut, mit diesen Befehlen kannst du schon mal alles wichtige mit Dateien machen.
Und das wird nun geübt!
Alles dann mal los: Dateien erzeugen, Verzeichnisse kopieren, Dateien in Verzeichnisse verschieben, Dateien im Verzeichnisse löschen, Verzeichnisse mit Abfrage löschen, Dateien erzeugen ... bis die Tastatur um Hilfe bettelt.

5 - Linux Dokumentation

man

Fast jeder Konsolen-Befehl hat sein Manual, sein Handbuch.
Aufgerufen wird es mit **man**:

```
[tux] ~$ man ls  #manual zu ls anzeigen
```

Mit Pfeil-Hoch/Runter kann man durch den Text gehen (wenn **less** der Betrachter ist).
Mit "/other" ENTER kannst du alle Wörter in denen "other" vorkommt im Text suchen.
Mit "q" kommst du wieder aus dem Manual heraus.

Du kannst die Manuals mit **apropos suchwort** durchsuchen, damit werden alle Hilfetexte die zum suchwort passen angezeigt:

```
[tux] ~$ apropos ln #Alle Hilfen zu ln anzeigen

...
ln (1)               - make links between files
sln (8)              - static ln
pbmtoln03 (1)        - convert protable bitmap to DEC LN03+ Sixel output
...
```

Die Zahl hinter dem Befehl kennzeichnet einen Themenbereich. Gibt es mehrere Bereiche zu einem Thema ,musst du die Nummer mitangeben:

```
[tux] ~$ man 1 ls  #manual zu ls, Bereich1 anzeigen
```

info

Ist der Nachfolger von **man**, aber noch nicht überall vorhanden
Aufruf:

```
[tux] ~$ info ls  #info zu ls anzeigen
```

Beenden mit "q"

The Linux Documentation Project

Das Linux Document Project war jahreland DIE Quelle für Linux-Doku. Mittlerweile sind dort die meisten Dokus veraltet und wahrscheinlich nicht mehr brauchbar. Mittlerweile hat sich das Wissen in Wikis, Foren und Blogs verschoben, kaum noch einer schreibt Doku in diesem Stil; auch viele private Linux-Seiten gibt es nicht mehr (tuxhausen ist ja auch nicht mehr ganz frisch ;-))

Trotzdem hier der Link zum Linux Documentation Project, da dort doch immer noch einige gute HowTo, Guides und FAQs liegen:
http://tldp.org/docs.html

Es gibt auch einige deutsche HowTOs die aber noch älter sind; diese findest du unter:
http://www.linuxhaven.de/dlhp/

RFC

In den RFC, Requests For Comments, steht wie das Internet funktioniert, alle Protokolle sind hier spezifiziert.
Diese Infos sind sehr speziell und nur für Fortgeschrittene, aber manchmal sehr nützlich. Im Original zu finden unter:
http://www.rfc-editor.org/
Aber besser aufgeteilt unter:
http://www.faqs.org/rfcs/index.html

Suchmaschinen

Die beste Möglichkeit etwas über zu finden ist wie fast immer ... Google.
Dort findest du fast immer etwas, aber versuche nicht nur die Web-Suche, sondern auch mal über die Groups.
Noch ein Tipp, auch ruhig mal die Ergebnisse auf den 2. oder 3. Seiten der Suche angucken ...

Und es gibt eine spezielle Suche für Linux:
http://www.google.de/linux

Wenn alles nicht hilfst ...

... dann schreib doch mal eine E-Mail an den Entwickler des Programms, die meisten sind recht freundlich und antworten zügig, ABER sehe das immer als **letzte Möglichkeit.** Denn meist werden dem Autor immer die selben Fragen gestellt und das obwohl diese bereits irgendwo im Web beantwortet stehen (darum FAQ!) !!
Viele Programme haben Wikis oder Foren auf der Homepage, frag lieber dort erstmal nach.

6 - Editoren

Eines der meistgenutzen Programme unter Linux ist ein Editor zum Verändern von Dateien. Da die Konfigurationen von Programmen zumeist in Textdateien enthalten sind, sind Editoren besonders wichtig.
Gerade bei den Editoren gibt es wahre Glaubenskriege, der eine hält den **vi** für den besten Editor, der nächste den **emacs** oder **mcedit** und das waren einige Textkonsolen-Editoren .... Ich gebe hier mal eine kleinen Einblick in die Editoren und zwar wie man mit ihnen den Inhalt einer Datei ändert.

Fangen wir mit dem urigsten an ....

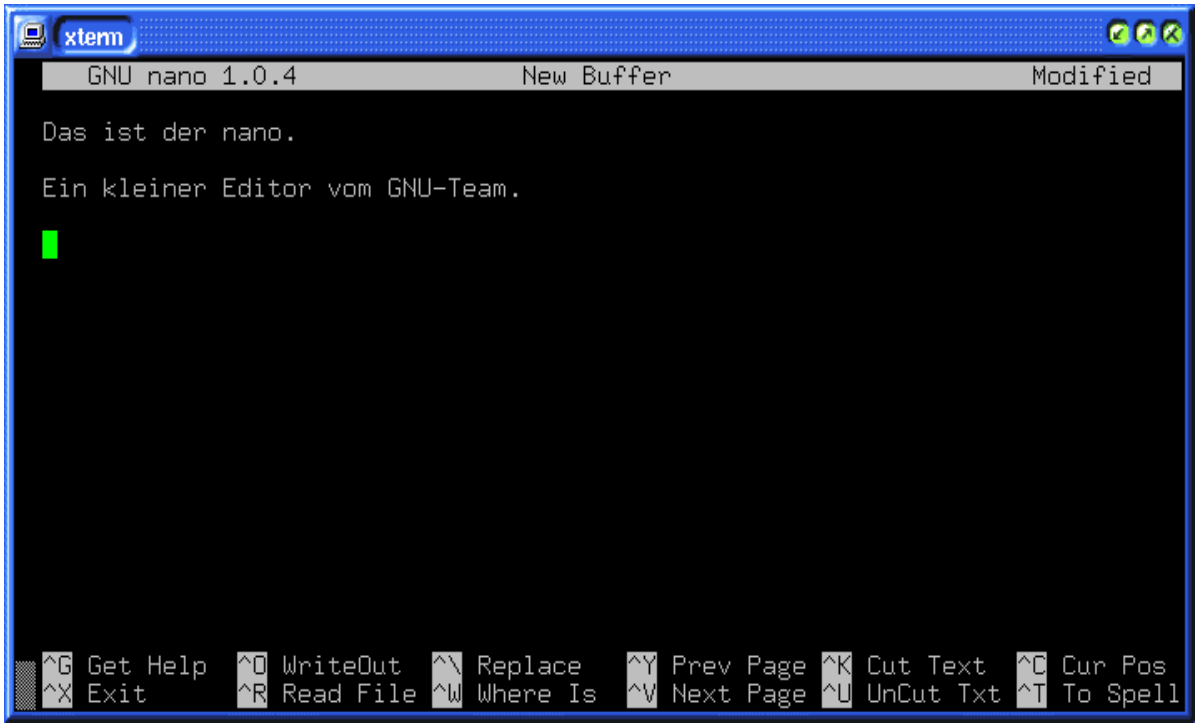
Textkonsolen Editoren

vi

Zum **vi** gibt es unter tuxhausen schon einen Artikel, deshalb verweise ich auf diesen:
tip_vi.html
Wenn du mehr über den **vi** erfahren willst guck dir mal Wolfgangs Artikel auf Pro-Linux an :
http://www.pro-linux.de/t_programme/vi_tut.html und kauf dir eine **vi** Tasse ...

GNU nano

Dieser Editor ist klein, einfach und nett ;-)) Er gehört zum GNU-Projekt und ist ein freier pico-Klon, ich nutze ihn selber gerne.
Das Nette ist das nano unten in der Leiste anzeigt, wie du welches Kommando aufrufst. Hier mal ein Bild von nano:



Die Befehle :

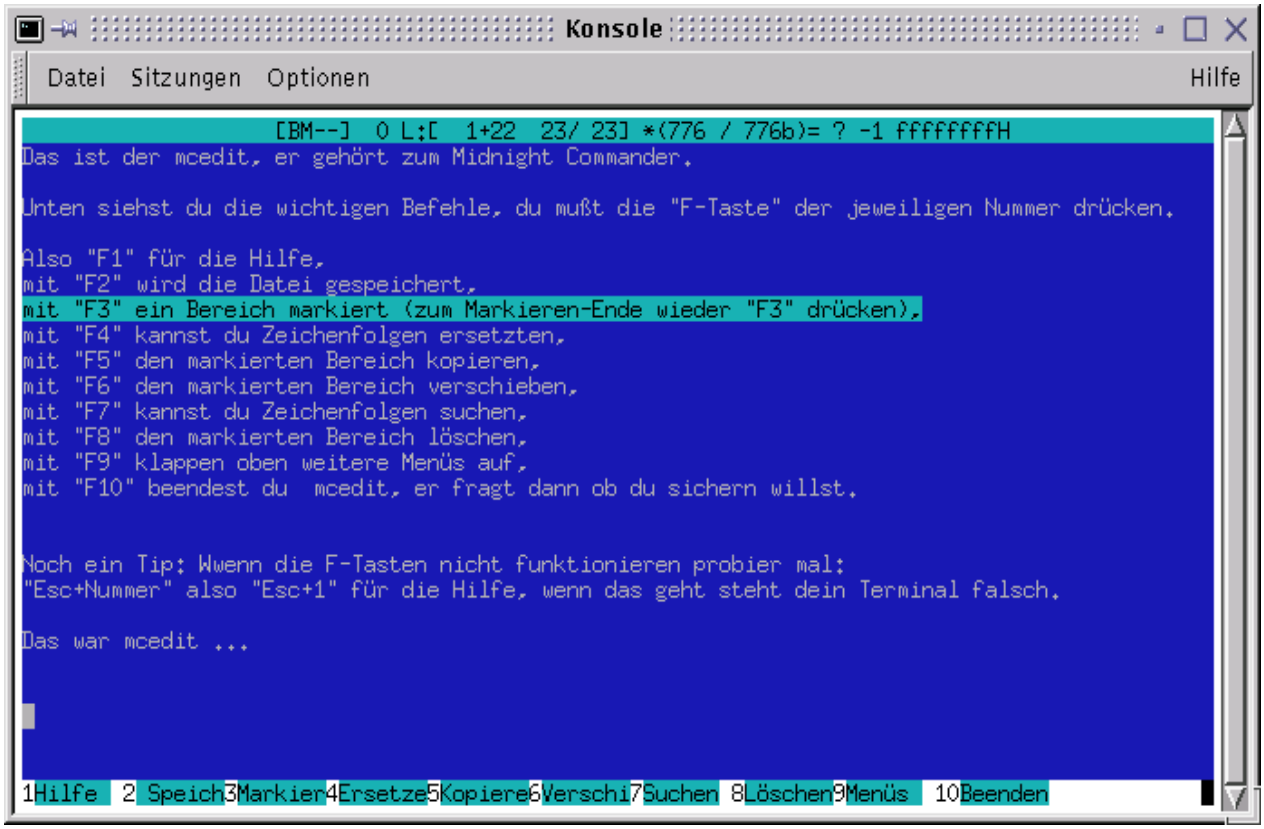
Das Zeichen ^ steht für die Taste "Strg".
Du kannst die Buchstaben klein oder groß schreiben:

Mit "Strg+O" schreibst du die Datei,
mit "Strg+R" liest du eine Datei ein,
mit "Strg+K" schneidest du eine Zeile aus,
mit "Strg+U" fügst die sie (evtl. mehrmals) wieder ein,
mit "Strg+X" beendest du pico, er fragt dann nach ob du sichern willst.

Und mit "Strg+G" bekommst du eine Hilfe !!
Wenn oben in der rechten Ecke "Modified" steht, hast du die Datei geändert ,aber noch nicht gesichert !

`mcedit`

Ein sehr einfacher und relativ mächtiger Editor, mein persönlicher Liebling. Der **`mcedit`** gehört zum Midnight Commander und ist eigentlich in jeder Distribution erhalten.
Hier mal ein Bild vom **`mcedit`**:



Auch hier noch mal die im Bild gegebene Erklärung:

Unten siehst du die wichtigen Befehle, du musst die "F-Taste" der jeweiligen Nummer drücken.

Also "F1" für die Hilfe,
mit "F2" wird die Datei gespeichert,
mit "F3" ein Bereich markiert (zum Markieren-Ende wieder "F3" drücken),
mit "F4" kannst du Zeichenfolgen ersetzten,
mit "F5" den markierten Bereich kopieren,
mit "F6" den markierten Bereich verschieben,
mit "F7" kannst du Zeichenfolgen suchen,

mit "F8" den markierten Bereich löschen,
mit "F9" klappen oben weitere Menüs auf,
mit "F10" beendest du mcedit, er fragt dann ob du sichern willst.

Noch ein Tipp: Wenn die F-Tasten nicht funktionieren, probier mal: "Esc+Nummer" also "Esc+1" für die Hilfe, wenn das geht, steht dein Terminal falsch.

emacs

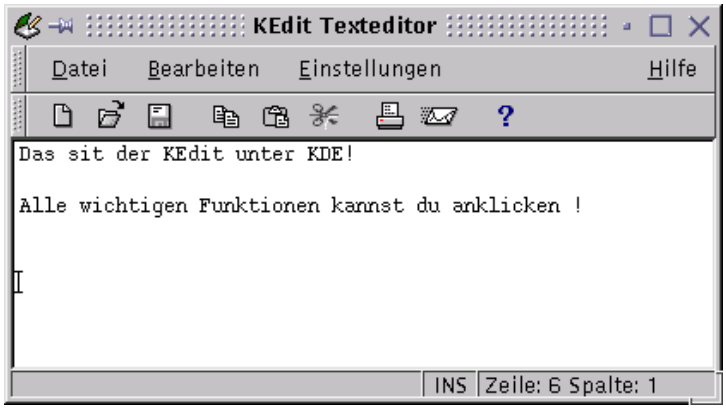
Der **emacs** ist wohl einer der mächtigsten Editoren und steht , wie der `vi`, wohl auf allen Unixes (und auch Windows) zur Verfügung. Ich persönlich habe ihn noch nie direkt benutzt (ja ja), und sage darum nichts dazu, außer: Es gibt ihn auch in einer X-Windows Version, der X-Emacs.
Wen du mehr wissen willst, guck mal in Kleine Einführung in den emacs-Editor

X-Window Texteditoren

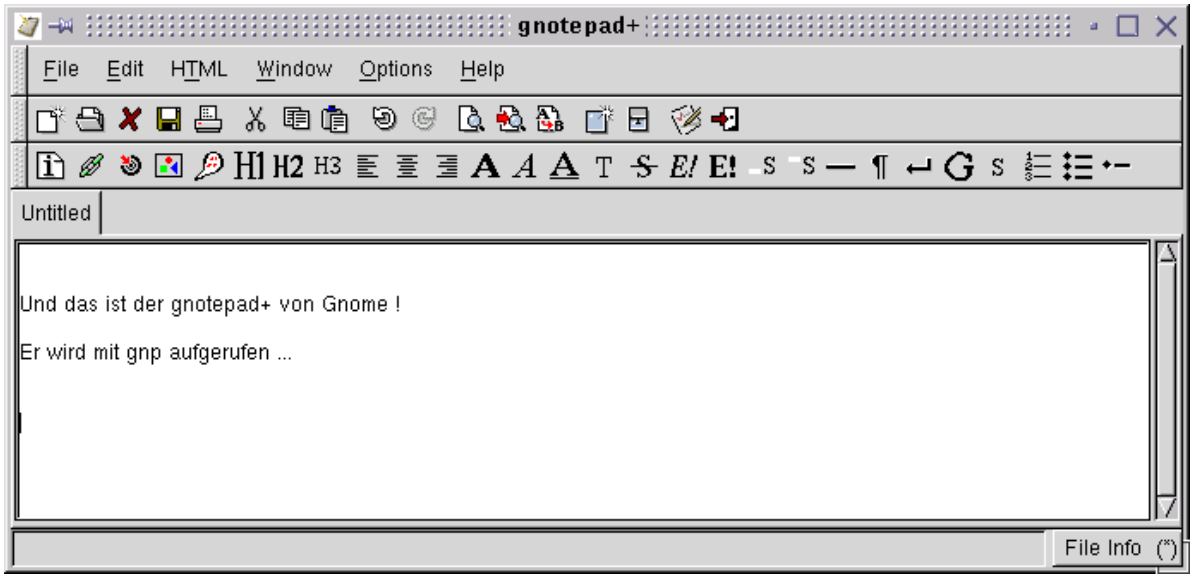
Wie auch an der Konsole gibt es wohl hunderte Editoren für X-Window.
Diese Programme brauche nicht zu beschreiben, da sie alle zum Klicken sind und deswegen sehr einfach zu bedienen (wenn du nicht umbedingt alle Feature ausnutzen willst).
Aber ein paar Bildchen können ja nicht schaden ...

KEdit

Läuft nicht nur unter KDE, aber sieht dort am besten aus.

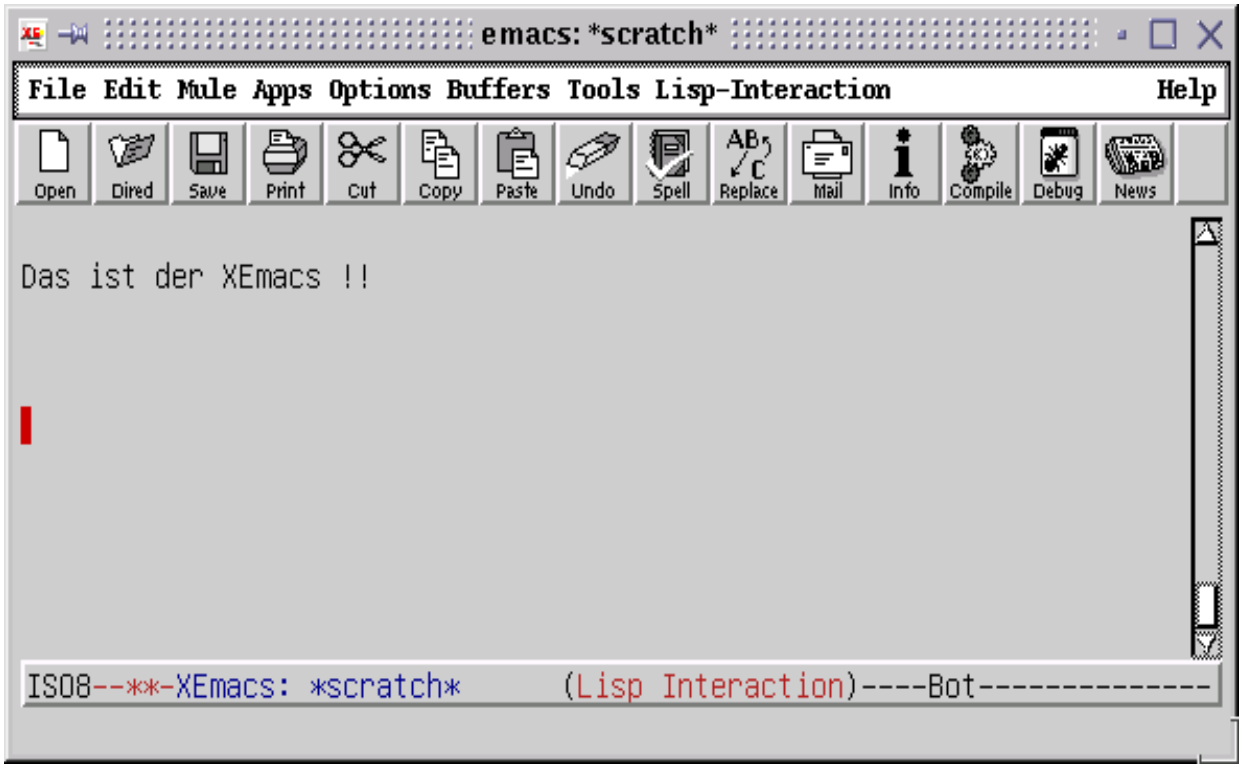


gnotepad+



XEmacs

Der mächtige Editor und Alleskönner:



7 - Mounten

Unter Mounten versteht man das Einbinden von Laufwerken in das Unix-Verzeichnis-System. Denk noch mal an das `/mnt` Verzeichnis, direkt unterhalb des Root-Verzeichnisses.

Laut dem Filesystem Hierarchy Standart sollen alle temporär gemounteten Laufwerke unter `/mnt` gemountet werden.
Alle Medien (CDROM, DVD, Floppy, Zip, ...) sollten unterhalb `/media` gemountet werden.
Du kannst aber nicht nur Geräte mounten, sondern auch Netzwerklaufwerke.

Voraussetzungen zum Mounten

Das Verzeichnis in das du etwas mounten willst, muss existieren und am besten leer sein. Ist es nicht leer, so siehst du deine Daten dort während des Mountens des Gerätes nicht mehr, danach aber wieder .
Desweiteren darf normal nur der Superuser root etwas mounten, aber das lässt sich ändern (schließlich ist das hier Linux).
Dann sollte noch das Filesystem des zu mountenden Gerät bekannt sein, hier mal die Wichtigsten:

- FAT - DOS und Windows
- FAT32 - ab Windows 95B (heißt aber unter Linux VFAT)
- NTFS - Windows NT, Windows 2000, Windows XP, Windows Server 2003
- SMB - Windows Netzlaufwerke
- ext2/ext3 - das meistbenutzte Linux-Dateisystem
- NFS - Linux Netzwerklaufwerke
- ISO9660 - CDROM
- HFS - Apple Filesystem

Und dein Kernel muss diese Dateisysteme auch mounten können !
Ob er das kann, erfährst du mit dem Befehl:

```
[tux] $ cat /proc/filesystems        #Anzeige der verfügbaren Filesysteme
```

Wenn das Filesystem dort auftaucht, hast du schon mal gewonnen ....

Mounten

Die (einfache) Syntax des Befehls **mount** heißt:

```
mount -t Dateisystem Gerät Mountpoint
```

Disketten

Disketten sind meist FAT, VFAT oder ext2 formatiert (ältere auch noch mit Minix).

```
[root] # mount -t vfat /dev/fd0 /media/floppy
#Floppy nach /media/floppy mounten
```

CDROM und DVD

Diese sind meist in ISO9660 formatiert:

```
[root] # mount -t iso9660 /dev/scd0 /media/cdrom   #erstes SCSI-CDROM nach
/media/cdrom mounten
...
mount: Blockorientiertes Gerät /dev/scd0 ist schreibgeschützt, es wird im Nur-Lese-
Modus gemountet.
....
```

oder

```
[root] # mount -t iso9660 /dev/hdc1 /media/cdrom   #IDE-CDROM nach /media/cdrom
mounten
...
mount: Blockorientiertes Gerät /dev/hdc1 ist schreibgeschützt, es wird im Nur-Lese-
Modus gemountet.
....
```

umounten

Und nun probier mal die CD wieder herauszubekommen ...
Das geht nicht, weil das Gerät noch gemountet ist.
Du denkst nun wahrscheinlich: Warum ist das so, ich weiß doch das ich nicht mehr auf die CD zugreifen will ?
Jetzt überleg mal wieviel Spaß es macht, im Netzwerk aus dem CD-Tower eine CD herauszunehmen von der gerade jemand anderes was kopieren möchte !

Die (einfache) Syntax des Befehls **`umount`** heißt:

```
umount Mountpoint
```

Also für unser CDROM:

```
[root] # umount /media/cdrom    #CDROM umounten
```

Jetzt kannst du die CD herausnehmen ;-))

Die Datei `/etc/fstab`

Damit du nicht jedes mal den ganzen Mount-Befehl eingeben mußt, gibt es die Datei `/etc/fstab`.
Diese sieht zum Beispiel so aus:

```
/dev/hda2       swap            swap    defaults                0 2
/dev/hda1       /               ext2    defaults                1 1
/dev/sda1       /mnt/windows/C  ntfs    ro,noauto,user          0 0
/dev/scd0       /cdrom          auto    ro,noauto,user,exec     0 0
/dev/fd0        /media/floppy   auto    noauto,user             0 0
proc            /proc           proc    defaults                0 0
```

Im Klartext nach Spalte:

1. das Gerät
2. der Mountpoint
3. der Dateisystem Typ, bei `auto` wird versucht, das Dateisystem selber herauszufinden
4. die Option
5. Information für das Programm **`dump`** (wird zur Zeit ignoriert)
6. Überprüfen des Dateisystems

Die Spalten 4 und 6 muss ich etwas näher erläutern:

In Spalte 4 kannst du verschieden Optionen zu diesem Mountpoint angeben, die wichtigsten:

`exec`	Programmausführung erlaubt, bei CDROM
`default`	Standard-Voreinstellung
`gid=NUMMER`	Gruppenzugehörigkeit der Dateien (wenn von root gemountet)
`noauto`	Nicht automatisch beim System-Start mounten
`ro`	Read-Only, Schreibschutz
`sync`	Schreibzugriffe nicht puffern, ist langsamer aber noch sicherer
`uid=NUMMER`	Benutzerzugehörigkeit der Dateien (wenn von root gemountet)
`user`	jeder darf diese Gerät mounten

In der 6. Spalte wird angegeben, ob und wie das Dateisystem geprüft werden soll:

`0`	Keine Überprüfung
`1`	Root-Dateisystem (/)
`2`	Alle anderen zu überprüfenden Dateisysteme

Möchtest du nun erreichen das du deine Floppy als normale Benutzer so mounten kannst:

```
[tux] $ mount /media/floppy
```

Solltest du als root folgendes in die `/etc/fstab` eintragen:

```
/dev/fd0   /media/floppy    auto  noauto,user  0  0
```

Noch einfacher geht es mit KDE oder gnome, dort stehen dir kleine Helferlein zur Verfügung oder die CDs/DVDs tauchen direkt auf dem Desktop auf :-)

8 - Bash

Die `bash` (bourne again shell) ist eine Shell, und eine Shell ist eine Schnitstelle zwischen dem Linux-System und dem Anwender. Unter DOS ist es die `command.com` oder neudeutsch "Eingabeaufforderung".
An der Shell gibst du deine Befehle ein und das System führt sie aus, meist startest du ein Programm.
Unter Linux/Unix gibt (natürlich) mehrere Shells, zum Beispiel: `ksh`, `csh`, `ash`, ...
Aber die Standard-Shell ist die `bash` und deswegen gehe ich nur auf sie ein. Für alle (anderen) Shells gibt es genug Manual-Seiten.

Wichtige Tastenkürzel

Gerade die `bash` verfügt über einige geniale Tastenkürzel, die du später nicht mehr missen willst:

<TAB>-Taste

Fangen wir mit dem praktischsten an: **Der <TAB>-Taste**
Gib mal genau das hier ein:

```
[tux] $ mce <TAB>        #mce wird durch die <TAB>-Taste zu mcedit
```

Das geht nicht nur mit Programm-Namen, sondern auch mit Dateien:

```
[tux] $ ls /ho<TAB> tu<TAB>        #Sollte zu ls /home/tux/ werden
```

Findet die `bash` mehrere mögliche Erweiterungen gibt sie einen Signalton aus. Drückst du dann noch mal auf <TAB>, werden dir alle weiteren Möglichkeiten angezeigt:

```
[tux] $ n <TAB> <TAB>        #Alle Programm die mit n beginnen werden ausgeben
...
       namei             netstat         nice                    nm86           nroff
       native2ascii   newaliases  nisdomainname  nmblookup   nslookup
       neqn              newgrp         nl                        nohup          nsmail
       netscape       newkrn         nm                     novell_print
 ...
```

Du gibst dann den nächsten Buchstaben ein und drückst wieder auf <TAB> .

Pfeil hoch/runter Taste

Mit den Pfeil hoch/runter Tasten kannst du deine letzten eingebenen Befehl "zurückholen":

```
[tux] $ ls
[tux] $ <Pfeil nach oben>            #Nun steht wieder ls da
```

Wenn du mehrmals den <Pfeil nach oben> drückst, gehst du deine letzten Befehle von unten (letzter) nach oben (erster) durch,
und mit der <Pfeil nach unten> entsprechend andersrum.

Noch ein kleiner Tip: mit dem Befehl **`history`** zeigt dir die `bash` die letzten Kommandos an.

"Strg + R"

Mit dieser Tasten-Kombination kannst du in deinen letzten Befehlen suchen. Das Ganze braucht aber etwas Erklärung.
Nach der Eingabe des ersten Zeichens schlägt dir die bash schon was vor, aber das stimmt meist wohl nicht. Du kannst nun,

- einfach die nächsten Buchstaben eintippen (egal was da jetzt steht)
- nochmal "Strg + R" drücken, dann wechselst du in den verschiedenen gefundenen Möglichtkeiten
- mit "TAB" die gefundene Zeile weiter editieren
- die gefundene Zeile mit "ENTER" ausführen

"Strg + L"

Damit kannst du den Bildschirm-Inhalt löschen

Es gibt noch einige weitere Tastenkürzel, die ich aber für nicht so wichtig halte ...

Umleitungen

Manchmal ist es nützlich sich ein Ergebnis in eine Datei schreiben zu lassen, anstatt auf den Bildschirm, das geht ganz einfach:

```
[tux] $ ls > inhalt            #Die Ausgaben von ls werden in die Datei
inhalt geschrieben
```

Existierte die Datei inhalt schon wird sie radikal überschrieben !! Existierte sie noch nicht, wird sie angelegt.

Möchtest du aber nun an eine Datei etwas anhängen solltest du zwei > benutzen:

```
[tux] $ ls /etc >> inhalt            #Die Ausgaben von ls werden an die Datei
inhalt angehangen
```

Allerdings kommt jetzt hier eine kleine Falle, denn unter Linux/Unix gibt es :

- Standardeingabe: Aus ihr lesen die Programme ihre Eingabe, normalerweise ist es die Tastatur
- Standardausgabe: Alle Programm-Ausgaben werden an die Standart-Ausgabe geschrieben, normalerweise der Bildschirm
- Standardfehler: Die Fehlermeldungen werden hier angezeigt, ist meist die Standardausgabe, also der Bildschirm

Mit unserer obigen Zeile haben wir **nur die Standardausgabe** in eine Datei geleitet, aber nicht der Standardfehler !!
Um die Fehlermeldung in eine Datei zu schreiben, mußt du **2>** schreiben und um die Standardausgabe und Standardfehler in eine Datei umzuleiten:

```
[tux] $ ls /etc >& inhalt            #Ausgaben und Fehler von ls werden an die Datei
inhalt geschrieben
```

Dann hätten wir noch einen seltenen Fall: Einige Programme erwarten immer Eingaben von der Standardeingabe. Dazu zählt auch das Programm **sort**, es sortiert den Text von der Standardeingabe. Das ist aber meistens unerwünscht, also leiten wir die Eingabe um:

```
[tux] $ sort <inhalt            #Der Inhalt der Datei inhalt wird
sortiert ausgegeben
```

Aber Achtung !! Du kannst nicht das Ergebnis von **sort** wieder nach inhalt schreiben, du würdest die Datei löschen !!

Pipes

Mit Hilfe einer Pipe gibst du die Ausgabe von Kommando1 an die Eingabe von Kommando2 weiter, etwas irretierend aber genial.
Dieser Befehl ist etwas blöd, da **ls** sowieso schon sortiert, aber egal:

```
[tux] $ ls | sort            #Ausgabe von ls wird von sort sortiert
```

Eigentlich einfach zu lesen:
Die Daten die **ls** ausgibt, werden an die Eingabe von **sort** umgeleitet und **sort** verarbeitet diese Daten,
man könnte nun noch eine Umleitung dranhängen:

```
[tux] $ ls | sort > SortierErgebnis.txt
```

Jetzt gibt **sort** noch seine Dateien in die Datei SortierErgebnis.txt aus. :-))

Eine nette Pipe sollte man noch erwähnen:

```
[tux] $ ls | tee inhalt            #Ausgabe in Datei und auf Bildschirm
```

Die Ausgabe von **ls** wird von **tee** auf den Bildschirm **und** in eine Datei geschrieben.

Kommandoausführung

Die Bash kennt verschiedene Arten der Programmausführung, die Wichtigste ist das Ausführen eines Programmes im Hintergrund.
Das wird oft bei Programmen oder Anweisungen benutzt die länger brauchen:

```
[tux] $ ls -R > inhalt_platte.txt &          #ls im Hintergrund ausführen
[1] 1301
```

Mit dem Kommando **ls -R > inhalt_platte.txt** werden alle Dateinamen unterhalb des aktuellen Verzeichnisses in die Datei inhalt_platte.txt geschrieben und zwar im Hintergrund, dafür steht das **&**.
Die bash teilt uns dann noch die Prozessnummer mit, mit dieser können wir das Programm evtl. abbrechen.

Hier noch ein Tipp: Sollte dir zu spät auffallen, das dieser Prozeß zu lange dauert, kannst du ihn mit "Strg+Z" unterbrechen und dann mit **bg** weiter im Hintergrund laufen lassen !

Die nachfolgende Tabelle zeigt weitere Arten der Programmausführung:

kommando1 && kommando2	kommando2 wird nur ausgeführt wenn kommando1 erfolgreich war
kommando1 || kommando2	kommando2 wird nur ausgeführt wenn kommando1 einen Fehler lieferte
kommando1 ; kommando2	kommandos werden nacheinander ausgeführt

Joker und Sonderzeichen

Die Joker-Funktion ist sehr einfach (ok, auch kein so tolles Beispiel ...) :

```
[tux] $ ls *.txt            #ls zeigt alles an was mit .txt endet
...
 SortierErgebnis.txt  inhalt_platte.txt  mcedit.txt  pico.txt  vi.txt
...
```

So folgendes passiert in unserem Befehl:
Die bash ersetzt *.txt durch alle Dateien die mit txt enden, wichtig ist, die bash ersetzt, nicht **ls** !
Du könntest auch noch das Fragezeichen (?) anstelle dem Stern einsetzen, das ? steht für ein einzelnes Zeichen. Es gibt noch weitere Möglichkeiten, aber ich finde das reicht, kannst ja bei Interesse mal in den Kofler gucken ...

Eine Sache noch:
Wenn du eine Datei mit dem Namen "Hallo ich bin ein Text .txt" anlegen willst, geht das so nicht. Denn ein Leerzeichen dient ja der Parameter-Trennung.
Also probier mal:

```
[tux] $ touch "Hallo ich bin ein Text .txt"            #Sonderzeichen
```

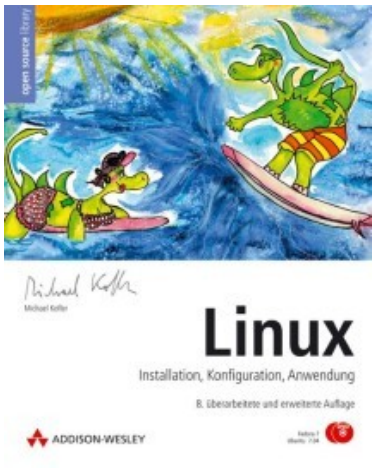
Und schon geht das !!

Auf tuxhausen habe ich noch einen interessanten Artikel zur `bash`: `bash` anpassen

9 - Übersicht der Kommandos

Diese Tabellen stammen aus dem Buch "Linux" von Michael Kofler.
Ich durfte diese nicht nur mit freundlicher Genehmigung des Autors benutzen,
sondern Michael schickte mir auch noch das LaTeX-Quell-Dokument !!!
Dafür vielen Dank !!

Auf den nächsten Seite des Buches folgt eine sehr gute Kommandoreferenz in der viele Beispiele gegeben werden.



Dateiverwaltung

`cat`	verbindet mehrere Dateien zu einer Gesamtdatei
`cd`	wechselt das aktuelle Verzeichnis
`cp`	kopiert Dateien
`ln`	stellt feste und symbolische Links zu Dateien her
`ls`	zeigt das Inhaltsverzeichnis an
`mkdir`	erzeugt ein neues Verzeichnis
`mv`	verschiebt Dateien bzw. ändert ihren Namen
`rm`	löscht Dateien
`rmdir`	löscht Verzeichnisse
`split`	zerlegt eine Datei in Teildateien mit vorgegebener Größe
`tee`	dupliziert die Standardeingabe

Dateien suchen

`find`	sucht Dateien nach Name, Datum, Größe etc.
`locate`	sucht Dateien in einer dafür vorbereiteten Datenbank
`whereis`	sucht Dateien in typischen `bin`-Verzeichnissen
`which`	durchsucht die `PATH`-Verzeichnisse nach Kommandos

Zugriff auf MS-DOS-Disketten

`fdformat`	formatiert eine Diskette (low level)
`mattrib`	verändert die Attribute der Datei
`mcd`	wechselt das aktuelle Verzeichnis
`mcopy`	kopiert Dateien von/nach Linux
`mdel`	löscht Dateien
`mdir`	zeigt das Inhaltsverzeichnis an
`mformat`	richtet ein DOS-Dateisystem ein
`mlabel`	verändert den Namen der Diskette
`mmd`	erzeugt ein neues Verzeichnis
`mrd`	löscht Verzeichnisse
`mread`	kopiert Dateien von DOS nach Linux
`mren`	ändert den Namen von Dateien
`mtype`	zeigt den Inhalt von Textdateien an
`mwrite`	kopiert Dateien von Linux nach DOS

Bearbeitung von Texten

`cat`	zeigt die Datei an bzw. vereint mehrere Texte
`csplit`	zerlegt den Text an vorgegebenen Stellen in Einzeldateien

`cut`	extrahiert Spalten aus jeder Zeile des Textes
`expand`	ersetzt Tabulator- durch Leerzeichen
`fold`	zerlegt lange Textzeilen in kürzere
`fromdos`	konvertiert DOS-Zeilenenden in das Linux-Format
`grep`	sucht Texte innerhalb der Datei
`head`	zeigt die ersten Zeilen der Datei an
`less`	zeigt Dateien seitenweise an (mit Rückwärtsbewegung)
`more`	zeigt Dateien seitenweise an
`paste`	vereint mehrere Texte zeilenweise
`recode`	konvertiert zwischen verschiedenen Zeichensätzen
`sed`	Stream-Editor (programmierbarer Editor)
`sort`	sortiert Dateien
`tac`	zeigt Dateien 'verkehrt' an (die letzte Zeile zuerst)
`tail`	zeigt das Ende der Datei an
`todos`	konvertiert Linux-Zeilenenden in das DOS-Format
`tr`	ersetzt vorgegebene Zeichen durch andere Zeichen
`uniq`	eliminiert mehrfach auftretende Zeilen in einer Textdatei
`zcat`	zeigt eine komprimierte Textdatei an
`zless`	zeigt eine komprimierte Textdatei an (auch rückwärts)
`zmore`	zeigt eine komprimierte Textdatei seitenweise an

Komprimieren und Archivieren von Dateien

`bunzip2`	dekomprimiert Dateien, die mit `bzip2` komprimiert wurden
`bzip2`	komprimiert Dateien; leistungsfähiger als `gzip`
`cpio`	überträgt Archivdateien zwischen unterschiedlichen Dateisystemen

`compress`	komprimiert Dateien
`gunzip`	dekomprimiert Dateien, die mit `gzip` komprimiert wurden
`gzip`	komprimiert Dateien; leistungsfähiger als `compress`
`mt`	steuert den Streamer (Vor- und Rückspulen etc.)
`tar`	vereint mehrere Dateien (und Verzeichnisse) in einer Datei
`uncompress`	dekomprimiert durch `compress` komprimierte Dateien

Prozessverwaltung

`bg`	setzt einen Prozess im Hintergrund fort
`fg`	setzt einen Prozess im Vordergrund fort
`halt`	beendet Linux und hält den Rechner an
`kill`	versendet Signale (meist zum vorzeitigen Beenden von Prozessen
`killall`	wie `kill`, der Prozess wird mit Namen genannt
`nice`	startet einen Prozess mit verringerter Priorität
`nohup`	startet einen 'unzerstörbaren' Prozess
`ps`	zeigt die Liste der laufenden Prozesse an
`ps`	wie `ps`, macht die Abhängigkeiten besser sichtbar
`reboot`	beendet Linux und startet den Rechner neu
`shutdown`	beendet Linux
`top`	zeigt alle fünf Sekunden eine Liste aller Prozesse an

Verwaltung von Benutzern und Gruppen

`adduser`	richtet einen neuen Benutzer ein (interaktiv)
`chsh`	verändert die Default-Shell nach dem Einloggen
`groups`	zeigt die Gruppen des aktuellen Benutzers an

`passwd` verändert das Passwort eines Benutzers

Administration des Dateisystems

`badblocks` testet, ob Datenträger (Disketten) defekte Sektoren enthalten

`dd` kopiert Datenblöcke zwischen Devices (low level)

`dumpe2fs` zeigt interne Informationen über ein ext2-Dateisystem an

`e2fsck` repariert ein ext2-Dateisystem

`fdformat` formatiert eine Diskette

`fdisk` partitioniert die Festplatte

`fsck` repariert ein Dateisystem (Frontend)

`mkfifo` erzeugt eine FIFO-Datei (eine benannte Pipe)

`mkfs` richtet ein Dateisystem ein

`mknod` erstellt Device-Dateien

`mkswap` richtet eine Datei oder eine Partition als Swap-Bereich ein

`mount` bindet ein Device (etwa eine Festplatte) in das Dateisystem ein

`swapoff` deaktiviert eine Swap-Datei oder -Partition

`swapon` aktiviert eine Swap-Datei oder -Partition

`tune2fs` verändert Systemparameter eines ext2-Dateisystems

Bildschirm und Terminal

`reset` stellt die Zeichensatzzuordnung wieder her

`restorefont` stellt den VGA-Zeichensatz wieder her

`restore-palette` stellt die VGA-Farbpalette wieder her

`setfont` verändert den VGA-Zeichensatz

`setterm` verändert diverse Terminaleinstellungen

Online-Hilfe

`apropos`	sucht Kommandos zu einem Thema
`info`	startet das `info`-System
`man`	zeigt die Beschreibung eines Kommandos an
`whatis`	zeigt eine Kurzbeschreibung (eine Zeile) eines Kommandos an

Sonstiges

`alias`	definiert eine Abkürzung
`cksum`	berechnet die CRC-Prüfsumme zu einer Datei
`date`	zeigt Datum und Uhrzeit an
`dmesg`	zeigt die Kernel-Meldungen des Bootvorgangs an
`expr`	führt einfache Integer-Berechnungen durch
`free`	zeigt den freien Speicherplatz (RAM und Swap-Speicher) an
`hash`	zeigt die Hash-Tabelle an
`ldd`	zeigt die erforderlichen Libraries für ein Programm an
`lpr`	druckt eine Datei aus
`printenv`	zeigt nur die Umgebungsvariablen an
`rdev`	verändert einige Bytes in der Kernel-Datei
`set`	zeigt alle der Shell bekannten Variablen an
`sum`	berechnet die Prüfsumme zu einer Datei
`tty`	zeigt den Device-Namen des aktuellen Terminals an
`type`	gibt den Typ eines Kommandos an (z.B.\ Shell-Kommando)
`unalias`	löscht eine Abkürzung
`uname`	zeigt den Betriebssystemnamen und die Versionsnummer an

10 - Software unter Linux installieren

Software bzw. Programme unter Linux zu installieren ist nicht sehr schwer, aber schwerer als unter Windows oder MacOS. Wenn du einige Dinge beachtest, wirdst du meinst keine Probleme haben.
Du brauchst dir auch nicht alles immer selber zu kompilieren (ok, schaden tut das nicht), die Zeiten sind vorbei.

Verschiedene Arten der Installation

Wir wären wohl nicht unter Linux, wenn es nur eine Art geben würde die Software zu installieren, oder ??
Also, hier mal eine kleine Auflistung wie du Software installieren kannst, von leicht zu schwer:

- **Installation-Programm deiner Distribution**
- **Installation-Programm des Software-Anbieters**
- **Paketmanager unter X oder der Console**
- **Quellcode kompilieren**

Vor allem haben die Distributionen in den letzten Jahren viel zu diesem Thema verbessert. Diese Anleitung kann evtl. recht flott veraltet sein !

Installation-Programm deiner Distribution

Die heutigen Distributionen werden mit Tausenden Paketen ausgeliefert. Ein Paket enthält meist das Programm sowie notwendige Dateien und einige Beispiel-Konfigurationen. Du suchst mit dem Installation-Programm deiner Distribution einfach das passende Paket und installierst es. Je nachdem wie gut die Distri-Leute waren, hast du direkt ein lauffähiges Programm, zum Teil mit Klick-Icon auf deinem Desktop.
Wenn du deine gesuchte Software als Paket deiner Distribution installieren kannst, installiere sie auf diesem Weg.
Das Problem ist dabei allerfings folgendes, die Version auf deiner Distribution ist meist schon älter und kann Fehler oder Sicherheitslöcher enthalten, oder einige Features noch nicht aufweisen, aber nur um mal zu gucken reicht es meist.
Wie du die Software installierst, schaue bitte in die Hilfe deiner Distribution, denn das ändert sich mit jeder Version etwas ...

Installation-Programm des Software-Anbieters

Einige Software-Buden bieten ihre Programme mittlerweile auch für Linux an, ich denke da an Adobe (Acrobat Reader), Borland (Kylix, JBuilder), ...
Diese Firmen versuchen es dem Anwender auch einfach zu machen und schreiben eigene Installations-Programme. Auf der einen Seite ist das ganz praktisch, aber der Nachteil ist, jeder bastelt wieder was Eigenes und es gibt keinen Eintrag im Paketmanager (dazu kommen wir gleich.)
Meist arbeiten diese Programme (noch) auf der Text-Console, aber einige naben auch schon schöne X-Programme geschrieben.
Auf der Web-Seite der Hersteller steht meist wie du die Software installierst, manchmal aber nicht...

Da beim Adobe Acrobat-Reader 4.05 (auf der deutschen Seite) nichts steht, nehme ich diesen mal als Beispiel:
Zuerst die Datei entpacken, du erkennst am Dateiende `.tar.gz`, das diese Datei mit `tar` und `gzip` gepackt wurde:

```
[tux] $ tar -xzf linux-ar-405.tar.gz        #Datei entpacken
```

Danach führst du im Verzeichnis `ILINXR.install` als root folgendes aus:

```
[root] ILINXR.install # ./INSTALL        #AcrobatReader installieren
```

Dann wird dir der Lizenzvertrag mit `less` angezeigt; beenden mit <q>, dann den Vertrag durch Eingabe von "accept" akzeptieren, nun den Installation-Pfad angeben (der Vorgeschlagene ist ganz ok), das Neuanlegen des Verzeichnisses bestätigen und das wars :

```
....
Do you want to create it now? [y] y
Installing platform independent files ... Done
Installing platform dependent files ... Done
...
```

Nun kannst du den AcrobatReader ausführen:

```
[tux] $ /usr/local/Acrobat4/bin/acroread        #AcrobatReader ausführen
```

ACHTUNG !! Adobe hat eine fiese Falle eingebaut: Es gibt noch einen anderen "acroread" (Reader/intellinux/bin), der ist falsch !!!
Da diese Eingabe etwas lang ist, macht du dir unter Gnome/KDE einen Button dafür, oder änderst den PATH, oder kopierst `acroread` nach `/usr/bin` ...

Paketmanager

Unter Linux gibt es noch ein kleines Problem beim Installieren von Programmen: Viele Programme setzen bestimmte Bibliotheken voraus damit sie laufen, der Sinn ist folgender: Wenn Programm A Bibliothek X installiert, braucht Programm B (welches auch Bibliothek X braucht) nicht auch noch Bibliothek X zu installieren. Das Dumme ist aber nun, weder Programm A noch B haben die Bibliothek dabei.
Nun darfst du dir selber die Bibliothek suchen und installieren...
Außerdem ist es schwer bei Tausenden von Programmen den Überblick zu behalten, welche Versionen man installiert hat, ob alle Bibliotheken vorhanden sind ,... Aus diesen wurden die Paketmanager entwickelt. Sie haben eine Datenbank in der steht welche Programme installiert sind, welche Bibliothek zu welchem Programm gehören muß, was beim Update überschrieben werden darf ,...

Unter Linux gibt es hauptsächlich 2 Paketmanager-Format: Das erste ist das DEB-Format von Debian, und das zweite ist das RPM (Red Hat Package Manager) von RedHat. Beide unterliegen der GPL und sind beinahe identisch, aber RPM ist verbreiteter, unter anderem benutzen es Red Hat, SuSE/Novell. DEB wird nur von Debian und Konsorten (Ubuntu) benutzt, ist aber nicht schlechter sondern zum Teil besser; zum Beispiel weiß DEB welche Pakete es installieren kann, ohne auf die CD zuzugreifen, da auch alle nicht Installierten in der Datenbank stehen.

Ich werde beide nur mal kurz ansprechen, das DEB-Format ist sowieso etwas schwierig zu verstehen. Noch ein Hinweis, Programme immer mit dem Paketmanager löschen, damit er nicht die Übersicht verliert.
Die ganze Operationen mit den Paketmanagern sollte man sinnvollerweise nur als root durchführen, du änderst ja dein System. Und noch ein Tip: Das Programm `kpackage` kann mit RPM und DEB (sowie SLACK, KISS, BSD) umgehen ;-))

RPM

Die wichtigsten Optionen von RPM mit einem Beispiel:

`--install bzw. -i`	installiert das Paket (mit Fortschrittsanzeige)	`rpm -ivh xpm-4.5-2.i386.rpm`
`--erase bzw. -e`	löscht ein Paket	`rpm -e xpm`
`--upgrade bzw. -U`	vorhandenes Paket aktualisieren	`rpm -U xpm-4.5-2.i386.rpm`
`--query bzw. -q`	Anfrage an die Paket-Datenbank alle installierten Pakete anzeigen = wozu gehört die Datei mail.rc =	`rpm -qa` `rpm -qf /etc/mail.rc`

DEB

Das Paketverwaltungssystem von Debian ist komplexer und leider auch schwieriger zu verstehen als RPM, ich hatte damit am Anfang große Probleme.
Erstmal vorweg, benutz nicht `deselect` wenn du es nicht brauchst, ich persönlich finde die Tastenbelegung sehr unglücklich, nimm lieber `apt` oder besser das aktuelle `aptitude`. Laut den Debianer-Maintaner sollten wir besser das neuere `aptitude` anstelle von `apt-get` verwenden; dann machen wir das auch so, obwohl dir `apt-get` immer noch zur Verfügung steht.
Wenn du Interesse an Debian hast, solltest du dir umbedingt das Buch Debian GNU/Linux von Peter H. Ganten aus dem Springer-Verlag zulegen, gerade wenn du an der Paketverwaltung verschweifelst. Am Anfang ist es schwer, aber dann ist es genial, dazu gleich mehr.
Die wichtigsten Optionen von DEB mit einem Beispiel:

`dpkg --install`	installiert/aktualisiert das Paket (als Datei)	`dpkg --install xpm-4.5-2.i386.deb`
`aptitude install`	installiert ein Paket aus der Datenbank	`aptitude install xpm`
`aptitude remove`	löscht ein Paket	`aptitude remove xpm`

aptitude update	Datenbank aktualisieren (neue Versionen)	`aptitude update`
aptitude upgrade	Alle Pakete aktualisieren	`aptitude -u upgrade`
apt-cache search	Nach Paketen suchen	`apt-cache search midnight`
dpkg	alle installierten Pakete anzeigen = wozu gehört die Datei mail.rc =	`dpkg --list` `dpkg -S /etc/mail.rc`

Nun noch eine sehr schöne Sache zu DEB: Wenn du in der Datei `/etc/apt/sources.list` die #-Einträge vor den http-Server entfernst, dann `aptitude update` und `aptitude upgrade` durchführst, ist dein System komplett auf dem neuesten Stand, eine dicke Leitung ins Internet solltest du aber schon haben ;-))

Quellcode kompilieren

Manchmal musst du Programme im Quellcode komplilieren und zwar dann wenn es keine fertigen Pakete für deine Distribution gibt.
Der Ablauf ist meistens der Selbe, einen kompletten beschreibe ich an dieser Stelle. Für diesen Zweck habe ich ein nützliches Programm ausgesucht, bei dem du gleich auch einen Patch einspielen kannst: gnupg. Gnupg ist ein freies PGP (Pretty Good Privacy), es dient zum sicheren Ver-/Entschlüsseln von Dateien. Es wird meist bei Emails angewandt.
Ich habe für dieses Beispiel die Dateien `gnupg-1.0.4.tar.gz` und `gnupg-1.0.4.security-patch1.diff` benutzt:

Auf der Download-Seite von gnupg sind die MD5 Checksummen angegeben, damit kannst du genau überprüfen ob deine gedownloadeten Dateien korrekt sind:

```
[tux] $ md5sum gnupg-1.0.4.tar.gz       #MD5 Summe ermitteln
bef2267bfe9b74a00906a78db34437f9 gnupg-1.0.4.tar.gz
```

Ok, passt; nun das Programm entpacken:

```
[tux] $ tar -xzf gnupg-1.0.4.tar.gz       #Entpacken
```

Das hätten wir auch, nun kommt der Patch rein. Bei den meisten Installationen brauchst du nicht noch zu patchen, aber so ist die Anleitung kompletter:

```
[tux] $ cp gnupg-1.0.4.security-patch1.diff gnupg-1.0.4
                          #Patch ins source-Verzeichnis kopieren
[tux] $ cd gnupg-1.0.4
[tux] gnupg-1.0.4 $ patch -p1 < gnupg-1.0.4.security-patch1.diff
                           #Patch ausführen
patching file g10/mainproc.c
patching file g10/plaintext.c
patching file g10/openfile.c
```

Ok, nun können wir endlich anfangen zu kompilieren.
Dafür gibt unter Linux meist diesen Dreisatz: `./configure`, `make`, `make install`. Für `make install` mußt du natürlich root sein, da das Programm irgendwohin installiert sein will. Ohne `make install` kannst du das fertig kompilierte Programm nur im Kompilier-Verzeichnis ausführen.

Ein wichtiger Tipp; ein Blick in die Datei **`INSTALL`** ist immer sehr angebracht ! Dort stehen Details zum Installieren drin.

Also los, zuerst ein `./configure`, hierbei wird überprüft ob du alle Bibliotheken und sonstnotwendige Software hast :

```
[tux] gnupg-1.0.4 $ ./configure
....
creating ./config.status
creating Makefile
creating intl/Makefile
creating po/Makefile.in
creating util/Makefile
creating mpi/Makefile
....
```

Dein Bildschirm ist nun voll und wenn irgendwo am Ende die makefiles erzeugt wurden, hat es geklappt. Ansonsten genau lesen was der Fehler war. Meist fehlt eine Bibliothek (lib), diese besorgst du dir dann (;-)) und installiert diese, dann führst du `./configure` erneut aus, bis es geht ...

Im nächsten Schritt, `make`, werfen wir endlich den Compiler an. Dieser Schritt kann bei lahmen Rechnern (< Pentium II) laaangeee dauern, nur so als Warnung :

```
[tux] gnupg-1.0.4 $ make        #kompilieren
```

Wenn `./configure` keine Fehler hatte, dürften jetzt auch keine entstanden sein. Wie ein Blick in `INSTALL` zeigte (und hast du reingeguckt?) können wir die Installation testen (make check ist eine Besonderheit von gpg und kommt meist nicht vor):

```
[tux] gnupg-1.0.4 $ make check
....
==================
All 24 tests passed
==================
....
```

Sehr schön, dein Programm (gpg) befindet sich übrigens im Verzeichnis `g10`. Damit es nun jeder nutzen kann, kommt `make install` zum Einsatz, als root:

```
[root] gnupg-1.0.4 # make install       #gnupg (gpg) fest installieren
```

Das war es, nun kannst du `gnupg` (`gpg`) benutzen und deine Dateien und Emails verschlüsseln ;-))
Wie schon erwähnt, die Installation hatte alles was beim Installieren passieren kann. Normalerweise kommst du mit dem Dreisatz `./configure`, `make`, `make install` aus.

11 - Benutzer, Gruppen und Zugriffsrechte unter Linux

Multi-User

Linux ist ein Mulituser-System, das heiß es können mehrere Benutzer gleichzeitig an Linux arbeiten. Das mag dich nun etwas verwundern, denn wie sollte so etwas gehen, bei nur einer Tastatur, Maus und Monitor? Ganz einfach, vor ca. 20 Jahren gab es nur große teure Computer und an diesem waren mehrere Terminals (meist nur eine Tastatur mit Monitor) angeschlossen und so arbeiteten alle Benutzer gleichzeitig an einem Rechner.
Aber dieses Verfahren ist auch einem normalen PC sehr gut, du kannst als Benutzer arbeiten und gleichzeitig als root irgendwas einstellen. Auf einem Server sind sowieso meist mehrere Benutzer eingeloggt und das ist auch Multi-User ;-))

So, damit nun nicht das totale Chaos ausbricht, und jeder bei dem anderen in den Dateien rumschreibt, werden für alle Dateien und Verzeichnisse Rechte fest gesetzt. Das heißt, User_1 kann in seine Datei schreiben, während User_2 diese nur lesen kann und User_3 sieht gar keine Datei. Daraus folgt, das sich jeder User an das System anmelden muß, denn sonst wüßte das System ja nicht wer du bist.

Jedes Unix (also auch Linux) hat einen speziellen Benutzer, den Systemverwalter. Dieser darf alles, also alles löschen, überall drauf zugreifen und alles kaputt machen. Dieser Benutzer ist root, im Windows-Umfeld ist es der Administrator. Wenn du nun denkst, he prima, ich mache alles als root, vergiß es, als root kannst du dein System *sehr schnell* zerstören. Also immer schön als normale Benutzer anmelden und als root den Rechner administrieren. Denn, root weiß was er tut ;-)

Gruppen

Eben hatten wir schon mal kurz die Rechte angesprochen die jeder Benutzer hat. Möchte man nun für mehrere Benutzer die selben Rechte einstellen, packt man diese in eine Gruppe und weist der Gruppe spezielle Rechte zu. Damit kannst du die Zugriffsrechte besser verteilen.
Damit die Sache noch etwas einfacher wird, kann jeder Benutzer in mehreren Gruppen Mitglied sein.

Benutzer anlegen und verwalten

Benutzer anlegen

Nur der Benutzer root darf neue Benutzer anlegen und alte löschen. Das Paßwort kann natürlich jeder Benutzer selber ändern.
Um einen Benutzer anzulegen gibt es wiedermal verschiedene Möglichkeiten:

- ein Tool deiner Distribution (z.B. yast bei SuSE)
- ein Programm unter X zum Beispiel kuser
- das Tool `adduser`, dieses fragt alle Eingaben interaktiv ab
- `useradd` ist eher für Skripte zu gebrauchen, da es alle Angaben als Parameter haben will

Benutze nun einfach mal `adduser` als root auf einer Konsole und legen Benutzer "testi" an:

```
[root] # adduser          #Neuen Benutzer anlegen
```

Noch ein Paar Tipps:

- einige Distributionen (RedHat, Debian) legen mit jedem neuen Benutzer eine neue Gruppe an, diese nennt sich wie der Benutzer, andere (SuSE) packen alle neuen Benutzer eine vorhandene Gruppe

- wird ein neues Home-Verzeichnis für den Benutzer erstellt, werden einige Beispiel-Dateien aus `/etc/skel` in das neue Home-Verzeichnis kopiert. Bei größeren Installationen evtl. einige Einstellung schon direkt in `/etc/skel` anpassen..
- Jeder Benutzer hat eine Standard-Einlog-Shell, meist sollte es wohl die `bash` sein. Für Benutzer die nur einen Daemon steuern (etwa http-run für den Apache) sollte die Standard-Einlog-Shell `/bin/false` sein, dadurch kann sich der Benutzer, oder besser der Cracker der das Passwort hat, nicht aktiv einloggen.

Benutzer verwalten

Tja, da zu gibt es das Benutzer-Anlege-Tool deiner Distri oder verschiedene (Konsolen) Programme:

`id`	zeigt Informationen über einen User an
`chsh`	ändert die Standard-Einlog-Shell des Users (Das kann nur root)
`passwd`	damit kann der Benutzer sein eigens Paßwort ändern

Wenn du schnell mehrere Dinge ändern willst, solltest du die Datei **`/etc/passwd`** von Hand editieren. Eine Zeile sieht dort so aus:

```
tux:x:501:100:Der kleine Tux:/home/tux:/bin/bash
```

Eigentlich ganz einfach zu lesen:

tux:	**x:**	**501:**	**100:**	**Der kleine Tux:**	**/home/tux:**	**/bin/bash**
der Benutzername	das Paßwort hier shadow-System	die Benutzer-Nummer	die Gruppen-Nummer	Beschreibung	Homeverzeichnis	Standard-Einlog-Shell

Hinweise:

- Das Paßwort steht normal verschlüsselt in der `/etc/passwd`, benutzt du aber (und das solltest du) das shadow-Paßwort-System steht dort nur ein "x" und das Paßwort selber steht verschlüsselt in der Datei `/etc/shadow`. Mit dem shadow-System kannst du auch Zugänge sperren (anstelle des verschlüsselten Paßwortes einen "*" eintragen), die Länge der Gültigkeit bestimmen ...

- Die Benutzer- und Gruppennummer sollte ich noch etwas erklären: Das Filesystem speichert zu jeder Datei nur diese Nummern ab, diese werden erst beim Ansehen mit `ls` oder sowas in die Namen gewandelt und diese Namen werden aus der `/etc/passwd` und `/etc/group` gelesen.

Gruppen

Zu den Gruppen gibt es weniger zu sagen. Im Wesentlichen reichen diese Befehle aus:

`groups`	zeigt die Gruppen des Benutzers an
`groupadd`	Gruppe hinzufügen. Als root
`groupdel`	Gruppe löschen. Als root
`groupmod`	Gruppen-Name oder Gruppen-Nummer ändern. Als root

Die Gruppen stehen in der Datei `/etc/group` und lassen sich dort schnell und einfach editieren. Hier mal ein Beispiel aus dieser Datei:

```
users:x:100:tux, tuxi
```

Und hier die Auflösung:

users:	**x:**	**100:**	**tux,tuxi**
der Gruppen-Name	das Paßwort	die Gruppen-Nummer	Mitglieder der Gruppe

Hinweise:

- Das Passwort ist auch hier mit dem shadow-System gesichert. Die dazu gehörige Datei ist `/etc/gshadow`.
- Jeder Benutzer kann Mitglied mehrere Gruppen sein. Einfach den Benutzer zu den Gruppen-Namen schreiben.

Rechte an Dateien und Verzeichnissen

Zuerst schauen wir uns mal eine Datei an:

```
[tux] $ touch datei
[tux] $ ls -l datei
....
-rw-r--r--    1 tux    users       1246 Mär 28 13:46 datei
....
```

Das heißt nun folgendes: Die Datei `datei` gehört dem Benutzer *tux* und der Gruppe *users* (zu der tux auch gehört). Wichtig sind aber die ersten Buchstaben, diese geben die Rechte der Datei an. Und zwar :

-	rw-	r--	r--
Typ: d = directory - = datei	Rechte des Eigentümers	Rechte der Gruppe	Rechte für alle anderen

Dabei sind:

r	read = lesen
w	write = schreiben, (auch löschen)
x	execute = ausführen des Befehls

So und nun zeige ich dir wie du die Rechte ändert, dazu musst du der Eigentümer der Datei sein (oder root), der Befehl lautet `chmod` (CHangeMODus) :

```
[tux] $ chmod u-w datei          #Rechte ändern
[tux] $ ls -l datei
....
-r--r--r--    1 tux    users       1246 Mär 28 13:46 datei
....
```

Und schon kann der Eigentümer nicht mehr in die Datei schreiben (ok, ist eine doofe Idee); die Syntax ist einfach: `chmod wer +/- was`:

- wer = bei wem werden die Rechte geändert; u=user (Eigentümer), g=group (Gruppe) o=other(alle anderen) oder a=all (user, group, other)

- +/- = + Recht setzen, - Recht wegnehmen
- was = Datei, für Verzeichnisse ein "-R" und alle drunterliegenden Verzeichnisse werden auch geändert

Für unser Beispiel von oben hieße das: Der User (`u`) bekommt das Schreibrecht(`w`) abgenommen (`-`).

Am besten probierst du ein Paar Sachen aus, dann ist es einfacher zu verstehen ...

Besonderheiten bei Verzeichnissen

Die Rechte haben bei Verzeichnissen eine etwas andere Bedeutung:

r	read = lesen => Verzeichnis ansehen (z.B.: mit `ls`)
w	write = schreiben, (auch löschen) => Dateien ins Verzeichnis schreiben
x	execute = in das Verzeichnis wechseln (z.B.: mit `cd`)

Auch hier, ist ausprobieren der beste Weg es zu verstehen ...

Sonderrechte

Neben den bereits bekannten Rechten (rwx) gibt es noch zwei besondere Rechte:

SUID = set user ID on execution	Das Programm wird beim Aufruf mit den Rechten des Eigentümers gestartet	`chmod u+s datei` `#SUID setzen`
GUID = set group ID on execution	Das Programm wird beim Aufruf mit den Rechten der Gruppe gestartet	`chmod g+s datei` `#GUID setzen`

ACHTUNG !! SUID und GUID können extrem gefährlich sein!! Wenn das Programm root gehört, wird jeder der es ausführt für das Programm root !!!
Möchtest du das auch andere Benutzer ein Programm mit root-Rechten ausführen können, solltest du das Programm `sudo` benutzen. Dort stellst du in einer zentralen Datei die Rechte der anderen Benutzer ein, das sorgt für mehr Überblick.

Eigentümer und Gruppe ändern

Manchmal möchte man nicht die Rechte der Datei ändern, sondern den Eigentümer oder die Gruppe. Dafür gibt es die Programm `chown` (CHange OWNner = Eigentühmer wechseln) und `chgrp` (CHange GRouP = Gruppe wechseln). Den Eigentümer wechseln kann nur root, während die Gruppe vom Eigentümer gewechselt werden kann. Die Syntax ist recht einfach:

```
[root] # chown neuer_owner datei  #datei gehört nun dem Benutzer neuer_owner


[tux] $ chgrp neue_gruppe datei   #datei gehört nun der Gruppe neue_gruppe
```

ACL - Access Control Lists

Diese Rechtevergabe ist der Standard unter den UNIXen, leider reicht das aber vielfach einfach nicht mehr. Manchmal musst du mehr als drei verschiedene Rechte vergeben. und dann?

Dann kommen Filesystem ins Spiel die ACL (Access Control Lists) beherschen, das gute alte ext2 kann das mit einem Patch auch, aber ich würde hierbei eher XFS oder JFS empfehlen.
Dazu mehr in diesem Tuxhausen-Artikel: Erweiterte Zugriffsrechte mit Access Control Lists

12 - Prozess-Verwaltung unter Linux

Wie du ja mittlerweile weißt (oder nicht) ist Linux ein Multitasking-Betriebssystem, es kann mehrere Programme quasi gleichzeitig laufen lassen. Linux gibt jedem Programm einen Sekunden-Bruchteil in dem es seine Arbeit verrichten muss, so kommt jedes Programm mal an die Reihe und der Benutzer denkt alle Programmen laufen gleichzeitig nebeneinander. Möchtest du mehrere Programm wirklich gleichzeitig laufen lassen, brauchst du auch mehrere Prozessoren.

Prozess-Status

Jedes laufende Programm besteht aus einem oder auch mehreren Prozessen und jeder Prozess hat einen Status. Damit das etwas klarer wird, gibt mal bitte als root den Befehl `ps` (Prozess Status) ein:

```
[root] # ps -fx          #Laufende root-Prozesse anzeigen
...
```

PID	TTY	STAT	TIME	COMMAND
1	?	S	0:04	init
2	?	SW	0:00	[kflushd]
3	?	SW	0:00	[kupdate]
4	?	SW	0:00	[kswapd]
201	?	S	0:00	/sbin/syslogd
203	?	S	0:00	/usr/sbin/sshd
207	?	S	0:00	/sbin/klogd -c 1
214	?	S	0:00	/usr/sbin/lpd
379	tty4	S	0:00	/sbin/mingetty tty4
1583	pts/3	S	0:00	bash
1855	pts/3	R	0:00	_ ps -f

Das sagt uns nun folgendes (ich nehme mal unseren `ps` Befehl):

1855	pts/3	R	0:00	_ ps -fx
Prozess -Nummer	Terminal	Prozess-Status	verbrauchte Rechenzeit in MM:SS	Datei-Name (+ Parameter)

Nun erstmal die wichtigsten Prozess-Statuse (?!):

R	running = läuft
S	sleeping = schläft und wartet auf Arbeit
Z	zombie = existiert nur kurz und verschwindet dann. Kann aber auch stehen bleiben, wenn der Eltern-Prozess gestorben ist. Hat keine Nachteile im System, kann ruhig komplett beendet werden

W	belegt keine Seiten im Speicher

Der Befehl `ps` kennt dutzende von Parameter, hier mal die Wichtigsten (die lassen sich auch gut kombinieren):

-a	Prozesse aller User anzeigen
-f	zeigt eine Baumansicht
-l	detailiertere Ausgabe
-u	zeigt den Besitzer des Prozesses an
-x	zeigt auch Prozesse an die zu keinem Terminal gehören

Du solltest auch mal den Befehl **`pstree`** ausprobieren. Was passiert siehst du schon ...
Ein weiteres nettes Programm zum Prozess überwachen ist **`top`**. Dieses solltest du in einem eigenen großen Fenster laufen lassen, da es sich alle 5sec aktualisiert. Beendet wird es mit der Taste <q>

Prozesse beenden

Kennst du die Prozess-Nummer, kannst du jeden Prozess (als root) beenden, bzw. killen (ja da heiß wirklich so). Der dazu benutze Befehl ist `kill`.
Du sagst ihm welchen Prozess er killen soll und er tut es:

```
[root] # kill 1522          #Prozess-Nummer 1522 wird beendet
```

Nun wird dem Prozess das SIGTERM-Signal gesendet und er wird sich gleich beenden....
Sollt der Befehl direkt gekillt werden, ohne ihm die Chance zu geben aufzuräumen oder ist es ein Zombie-Prozess, kannst du folgendes eingeben:

```
[root] # kill -KILL 1522          #Prozess-Nummer 1522 wird
SOFORT beendet
```

Das solltest du aber nur im Notfall machen !!!

Einigen Prozessen kann auch ein `-HUP` gesendet werden, damit sie ihre Konfiguration neu einlesen. Das steht dann in der `man`-Pages des Programmes.

Prozess-Priorität ändern

Jeder Prozess besitzt eine bestimmte Priorität bzw. Rechenzeit. Diese Priorität verteilt Linux selber und im Normalfall haben alle Programme die Priorität 0. Die Priorität verteilt sich von 20 bis -20: 20 ist ganz niedrig und -20 ganz hoch.
Im Normalfall solltest du an der Priorität nichts ändern, manchmal kann es nützlich.

Ein Beispiel: Du möchtest im Hintergrund ein Backup-Programm laufen lassen, möchtest aber beim Arbeiten nicht groß gestört werden. Du kannst das Backup-Programm mit einer niedrigen Priorität starten:

```
tux $ nice -n 10 backup.sh #Programm backup.sh mit der niedrigen Priorität 10 starten
```

Anzeigen kannst du die Priorität übrigens bei `ps` mit dem Parameter -l, oder mit top (Zeile NI).

Mit dem Befehl `renice` kann root (sonst keiner) die Priorität eines laufenden Prozesses ändern. Auch darf nur root eine höhere Priorität als 0 erteilen.

13 - Init und Runlevels

Systemstart durch Init-V

Direkt nach dem Kernel-Start wird unter Unix das Programm `/sbin/init` gestartet. Du siehst ihn als ersten Eintrag wenn du `ps afx` aufrufst. Du kannst dem Init-Prozess schon beim lilo/grub-Start Parameter übergeben, dazu später mehr.
Danach führt Init eine Reihe von Skripten aus, diese starten dann diverse Dienste oder konfigurieren deinen Rechner. Das richtig Interessante sind aber die Runlevels. Mit diesen Runlevels kannst du deinen Rechner in verschiedenen Konfigurationen hochfahren, zum Beispiel mit oder ohne Netzwerk.

Die Runlevels werden mit Nummer belegt, leider hat fast jede Distri eine andere Vorstellung davon. Aber es besteht Hoffnung, und zwar durch die Linux Standart Base, damit versuchen die meisten Distributoren eine einheitlich Basis für Linux zu schaffen.
Also, laut LSB sind die Runlevels so definiert:

0	System anhalten
1	Single User Mode: Nur ein Benutzer kann arbeiten, meistens root. Es sollten nur die wichtigsten Dienste gestartet sein
2	Mulituser, no network: Es können mehrere Benutzer arbeiten, ohne Netzwerk-Exports (NFS) (multiuser with no network services exported)
3	Normal, Multiuser: Normaler Modus
4	Reserviert: normale Benutzung, Multiuser
5	Multiuser mit X-Anmeldung: Es erscheint der X-Server zur Benutzer-Anmeldung
6	Reboot: Rechner wird neugestartet

Deinen aktuellen Runlevel erfährst du als root mit dem Befehl runlevel:

```
root # runlevel            #Der aktuelle Runlevel
```

Wenn du eine von diesen Nummern beim Start des lilo übergibst wird dieser Runlevel gestartet, zum Beispiel:

```
lilo: linux 1

oder
lilo: linux single
```

startet Linux im Runlevel 1 = single Modus. Kann es große Hilfe sein, wenn dein Netzwerk klemmt ...

/etc/inittab

Das ist die zentrale Konfigurationsdatei des Init-Prozeßesses, in dieser Datei steht auch drin welcher Runlevel was bei deiner Distri macht. Ein weiterer interessanter Punkt ist "ca::ctrlaltdel:/sbin/shutdown -t3 -r now" hiermit wird festgelegt was passiert wenn du auf die Tasten "Strg-Alt-Entf" (der Affengriff) drückst; hier wird ein schneller Neustart ausgeführt.
Der Rest in der Datei ist was für Fortgeschritte, daran rumzuspielen kann schwere Folgen haben !!

Startskripte

Die Startskripte dienen dazu die einzelnen Dienste zu starten oder zu beenden, sie sind (meist) bash-Programme.
So, damit der `init` nun weiß was tun soll, gibt verschiedene Verzeichnisse und zwar eines für jeden Runlevel. Diese Verzeichnisse stehen entweder direkt in `/etc/` (Debian) oder unter `/etc/rc.d/` (Mandrake, RedHat) oder unter `/sbin/init.d` (SuSE) und heißen `rc0.d, rc1.d, rc2.d ...` entsprechend dem Runlevel.

Ein solches Verzeichnis sieht in etwa so aus:

K15numlock
K20bootparamd
K20nfslock
K20rstatd
K20rusersd
.....
S09sound
S10network
S10usb
S20random
S30syslog

Dabei gilt nun folgendes:

- Der **Anfangsbuchstabe** gibt an ob es ein Start- oder ein Stopskript ist. Ist es ein Start-Skript (S wie Start) wird dem Skript der Parameter `start` übergeben, bei einem Stop-Skript (K wie Kill) der Parameter `stop`
- Die **Nummer** gibt die Reihenfolge an in der die Skripte gestartet werden
- Normalerweile sind alle Einträge Links, damit kann das Skript in verschiedenen Runlevels verwendet werden
- Zuerst werden die Stop-Skripte ausgeführt, dann die Start-Skripte

Anlegen eines eigenen Start-Skript

Normalerweile sollte es eine Datei namens "skeleton" im Skript-Verzeichnis geben, diese kannst du dein Skript nehmen, wenn es nicht existiert, einfach ein anderes Skript kopieren und dann bearbeiten
Unser Skript sollte folgendes machen: Wenn der Runlevel 2 erreicht ist, soll es 3mal piepen, und wird der Runlevel verlassen 1mal piepen.

Damit sieht unser Skript "piep" nun so aus (keine Angst wenn du es nicht verstehst, die `bash`-Syntax hatten wir im Kurs noch nicht, aber allzu schwer ist es nicht) :

```
#!/bin/bash

# Mit Parameter start piept der Rechner 3mal
# bei stop 1mal piepen

case "$1" in    #$1 ist der erste Parameter von piep

  start)        #wenn der erste Parameter start ist, also: piep start

        #dreimal piepen
        echo "Runlevel 2 erreicht "

        echo -n -e \\a   #piepen
        sleep 1          # eine Sekunde warten (schlafen)
        echo -n -e \\a   #piepen
        sleep 1          # eine Sekunde warten (schlafen)
        echo -n -e \\a   #piepen
        sleep 1          # eine Sekunde warten (schlafen)
        ;;

  stop)         #wenn der erste Parameter stop ist, also: piep stop

        #einmal piepen
        echo "Runlevel 2 beendet "

        echo -n -e \\a   #piepen
         ;;

  *)            #Anzeige bei allen anderen oder gar keinen Parametern
        echo "Usage: piep {start|stop}"
        exit 1
        ;;
esac

exit 0          #return Status fuer folgendes Programm
```

So dieses Skript nun in das Verzeichnis `rc2.d` als `S99piep` und als `K99piep` linken, und das war es schon :-)

Den Runlevel kannst du übrigens wechseln in dem es dem `init` sagst: `init 2` und es geht ab in Runlevel 2.

14 - bash-Skripte

bash-Skripte sind eine schöne Sache um immer wieder kommende Befehle zu automatisieren. Wir hatten im letzen Teil des Kurses schon ein kleines Skript: "piep". Es diente dazu, bei Eintritt und Verlassen des Runlevels kurz mal zu piepen.

Ein bash-Skript ist quasi eine Batch-Datei unter DOS, aber leistungsfähiger ;-)). Unter Windows ist es mit der PowerShell vergleichbar.

Hallo Tux

Ok, fangen wir mal an, mit dem üblichen "Hallo,Tux" Programm (oder so):

- Zuerst schreibst du mit einem beliebigen Text-Editor die Datei "hallo.sh". Die hat diesen Inhalt:
```
echo "Hallo Tux!"
```
- Damit du dein Skript auch ausführen kannst mußt du erst noch schnell die Rechte ändern:
```
[tux]$ chmod u+x hallo.sh
```
- Und nun die Datei im aktuellen Verzeichnis ausführen:
```
[tux]$ ./hallo.sh
Hallo Tux!
```

Das war es schon. Auf deinem Monitor solltest du nun "Hallo Tux!" sehen.

Jetz mit etwas mehr Anspruch:

```
#!/bin/bash
#Programm hallo2.sh

echo "Wie heißt du ?"
read Name

echo "Hallo $Name"
```

In der ersten Zeile sagst du der bash, dass dieses Skript von /bin/bash ausgeführt werden soll. Das ist eigentlich nicht umbedingt nötig, aber eleganter und kann Fehler vermeiden helfen.
In der zweiten Zeile steht ein Kommentar, dass siehst du an dem "#", dieses leitet fast immer einen Kommentar. Warum fast immer ?
Na guck noch mal in die erste Zeile, was es dort im Zusammenhang mit dem "!" heißt ...
In der dritten Zeile geben wie einen einfachen Text aus, und in der vierten Zeile lesen wir mit "read" eine Variable (ein Variable ist eine Art Speicher) ein, hier "Name".
In der letzen Zeile geben wir das Hallo mit dem eben eingegebenen Namen aus. Wird vor einer Variablen ein "$" geschrieben, wird der Inhalt der Variable ausgeben, und das ist der Name den du eben eingehämmert hast.

Parameterübergabe

Jetzt schreiben wir ein Programm, das direkt Parameter beim Kommandoaufruf benutzt.
Erst mal das Programm:

```
#!/bin/bash
#Programm hallo3.sh

echo "Hallo $1"
```

Dieses Programm rufst du nun so auf:

```
[tux]$ ./hallo.sh vorname
Hallo vorname
```

wobei vorname natürlich dein Vorname ist ...
Darauf sollte dein Programm "Hallo vorname" ausgeben.

If-Else Anweisung

Eine If-Else Anweisung läßt sich ungefähr mit "Wenn ... dann" übersetzen. Hier siehst du schon wofür du diese brauchst:
Du kannst einfach auf bestimmte Dinge abfragen.

Am klarsten wird das an einem Beispiel-Programm :

```
#!/bin/bash
#Programm if.sh

if [ "$1" = "tux" ]; then
   echo "Hallo Pinguin"
   #Wenn der erste Parameter tux ist, schreibe Hallo Pinguin

else
    echo "Hallo $1"
    #Sonst schreibe Hallo Parameter
fi

exit 0
```

Einfach mal ausprobieren mit:

```
[tux]$ ./if.sh tux
Hallo Pinguin
```

und danach:

```
[tux]$ ./if.sh markus
Hallo markus
```

Damit ist die If-Else Anweisung klar, oder?

Und jetzt mal die Syntax:
Die gesamte If-Else Anweisung beginnt mit dem if und endet mit fi (fi = if umgedreht). Alles dazwischen gehört zur If-Abfrage.
Die wichtigste Zeile:

```
if [ "$1" = "tux" ]; then
```

Hier sagst du:
Wenn (if) der erste Parameter ("$1") gleich (=) Tux ("tux") ist, dann (then)...
Danach schreibst du was passieren soll wenn die Anweisung zutrifft (echo "Hallo Pinguin").

Ab der Zeile mit `else` steht was passieren soll, wenn `if` nicht paßt (echo "Hallo $1").
In der letzen Zeile steht ein `exit 0`. Damit übergibst du dem nachfolgenden Programm einen sogenannten Rückgabe-Parameter. Damit kannst du dem nachfolgenden Programm anzeigen, ob das Skript erfolgreich beendet wurde oder ein Fehler aufgetreten ist. Übrigens, viele Unix/Linux-Programme nutzen diese Option.

Achtung!! Die Syntax der If-Else Anweisung muß genauso geschrieben werden (die Leerzeichen sind EXTREM wichtig !), sonst geht deine If-Abfrage nicht.

Noch ein Tipp: Willst du Zahlen vergleichen mit == , >= , < ; mußt du -eq (== equal), -ge (>= greater equal), lt (< less than) nehmen.

Case Anweisung

If-Else ist ja ganz nett, aber wenn du auf mehrere verschieden Parameter abfragen willst, wird es schon etwas komplexer. Hier hilft dir die Case Anweisung.
Man kann sie so beschreiben: Wenn 1 dann das, Wenn 2 dann dieses, Wenn 3 dann dasda, Wenn nichts davon dann...

Auch hier wieder ein erklärendes Beispiel:

```
#!/bin/bash
#Programm case.sh

case "$1" in
     tux)
            echo "Hallo Pinguin"
                 #Wenn Parameter 1 = tux
            ;;
     markus)
            echo "Hallo, alter Sack ;-)"
                 #Wenn Parameter 1 = markus
            ;;
     *)
             echo "Hast du keinen richtigen Namen ? ;-)"
                  #Wenn vorher nichts passte
             ;;
esac

exit 0
```

Einfach mal ausprobieren mit:

```
[tux]$ ./case.sh tux
Hallo Pinguin
```

und dann:

```
[tux]$ ./case.sh markus
Hallo, alter Sack ;-)
```

und danach:

```
[tux]$ ./case.sh dave
Hast du keinen richtigen Namen ? ;-)
```

Eigentlich ist die damit Syntax schon fast klar:
Zuerst sagt du worauf die Case-Anweisung reagieren soll: Hier auf Parameter 1 also `case "$1" in`.
Danach wird der Reihe nach abgeklappert ob eine Bedingung paßt. Unsere erste Anweisung beschreibe ich mal genauer:
Zuerst steht in der Anweisung : `tux)`, und das heißt: wenn der erste Parameter "tux" ist,
dann schreibe "Hallo Pinguin" (echo "Hallo Pinguin").
Wichtig ist die Anweisung mit " ;; " abzuschließen !!
Mit der letzen Anweisung (*)) wird alles abgefangen, was noch übrig ist, quasi eine default-Anwendung.

---

while-Schleifen

While-Schleifen laufen gewisse Anweisung solange durch, solange die Bedingung erfüllt ist.
Das kann man gebrauchen, wenn man die Zahlen 1-10 ausgeben will, die Schleife muß solange laufen, bis der Zähler nicht kleinergleich als 10 ist (immer dran denken: solange die Bedingung erfüllt ist).

Auch hier wieder ein Beispiel:

```
#!/bin/bash
#Programm while.sh

count=0           #Zähler auf Null setzen

#So lange durchlaufen  bis count nicht mehr unter 10 ist
while [ $count -le 10 ]
      do
        echo $count         #Zähler ausgeben
        count=$[$count+1]   #Zähler um eins erhöhen
      done

exit 0
```

Und einfach ausführen: Jetzt werden die Zahlen von 0 bis 10 hochgezählt:

```
[tux]$ ./while.sh
0
1
2
3
4
5
6
7
8
9
10
```

Leider ist die Syntax hier etwas schwieriger:
Zuerst setzen wir unseren Zähler count auf 0: `count=0`
Jetzt kommt die while-Bedingung: `while [ $count -le 10 ]` ; das dürfte dir zum Teil schon von der If-Else Anweisung bekannt vorkommen. Also, wir setzen hier als Bedingung count kleinergeich 10 ($count -le 10).
In der `do`-Anweisung steht drin, was ausgeführt wenn die Bedingung erfüllt ist. Wir geben hier den Wert des Zähler aus (echo $count) und erhöhen den Zähler um eins (count=$[$count+1]).

for-Schleifen

Die for-Schleife verhält sich unter der `bash` anders als unter C, und deswegen spreche ich sie nur ganz kurz hier an.

Hier ein Beispielprogramm:

```
#!/bin/bash
#Programm for.sh

for i in "Hallo du" da "am Moni" t or
    do
      echo $i
    done

exit 0
```

Einfach ausführen und das Ergebnis bitte erst mal selber auswerten:

```
[tux]$ ./for.sh
Hallo du
da
am Moni
t
or
```

For-Schleifen arbeiten der Reihe nach die angegebene Liste ab. Für Skripte ist das kaum zu gebrauchen aber für Dateioperationen ganz nützlich :

```
[tux]$ for i in *.sh; do cp $i $i.bak; done
```

Damit wird von alle Dateien im aktuellen Verzeichnis die auf .sh enden eine Kopie erstellt. Diese Kopie bekommt am Ende ein .bak angehängt.

Dialog

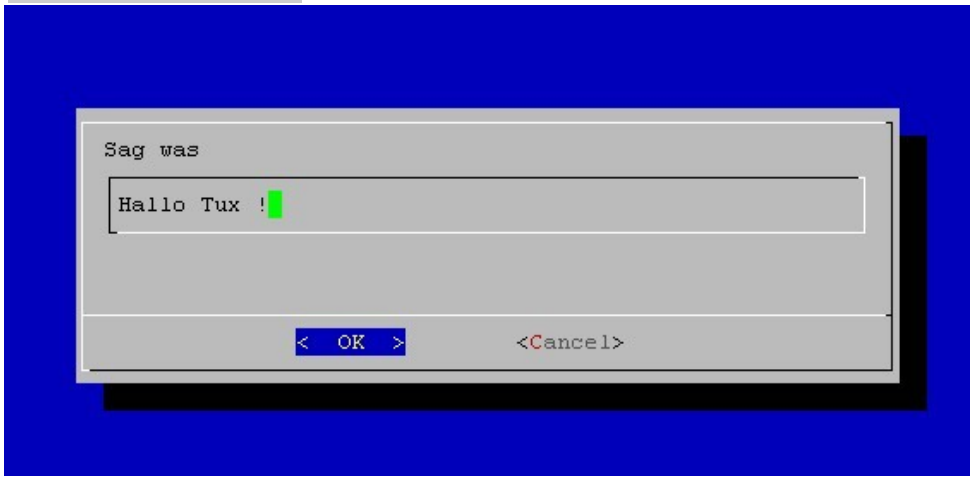
Zum Ende kommen wir zur grafischen Programmierung der `bash` (naja, es ist bunt und hat Felder). Das kann die `bash` nicht selber dafür brauchen wir einen kleinen Helfer. Dieser heißt "dialog".

Hier wieder mal ein Beipiel (ist das letzte auf dieser Seite):

```
#!/bin/bash
#Programm dialog.sh

#Eingabe Fenster mit dialog erzeugen
dialog --clear --inputbox "Sag was" 10 60 2> eingabe.tmp

clear        #Bildschirm löschen

#Eingabe einlesen und ausgeben
echo "Du sagtest: `cat eingabe.tmp`"

exit 0
```

Und ausprobieren ... :

```
[tux]$ ./dialog.sh
```




```
Du sagtest: Hallo Tux !
[tux]$
```

Das Wichtigste ist, dass du den eingegeben Text von dialog nicht direkt nutzen kannst, du mußt ihn in eine Datei schreiben (2> eingabe.tmp).
Dann lesen wir diese Datei aus (`cat eingabe.tmp`) und geben den Text aus. Hier sind die Anführungszeichen sehr wichtig (es sind die mit Shift+TasteNebenBackspace).

So das war ein kleiner Auflug in die `bash` Programmierung, wenn du selber Skripte schreiben willst, dann tu es :-))
Ein kleiner Tipp: Man kann viel lernen wenn man sich fertige Skripte ansieht, zum Beispiel die Startskripte in /etc/init.d.

15 - Netzwerk

Linux ist im Netzwerk geboren und fühlt sich dort auch am Wohlsten (Tux in der Antarktis).
Ich möchte hier mal ganz kurz beschreiben wie ein Netzwerk funktioniert, und wie du es per Hand einrichtest. Und genau hier kommt schon das erste Problem: Mittlerweile haben die meisten Distributionen eine gute Hardwareerkennung und haben deine Karte wohl schon eingerichtet. Für den Kurs bei Elzet80 habe ich die Treiber herausgeworfen. Das brauchst du zuhause nicht zu machen, aber ich denke es nicht schädlich zu sehen wie du eine Karte von Hand einrichtest.
Zuerst erkläre ich aber mal ein bißchen wie das Ganze so funktioniert, aber nur Ethernet (das haben eh die meisten).

Eine ganz kleine Einführung

Wirklich nur ganz klein, sonst wäre es ein eigener Kurs.
Ich nehme mir hier mal eine typische Situation und beschreibe (grob) was dann technisch passiert.
Aus technischer Sicht ist nicht alles hier genau genug beschrieben, aber das würde den Rahmen locker sprengen.

ping

Wenn du an der Konsole folgendes eingibst:

```
[tux]$ ping 192.168.15.66
```

dann soll dein Rechner versuchen, den Rechner mit der IP-Adresse 192.168.15.66 im lokalen Netz anzupingen.

Was ist denn eine IP-Adresse ?? Und ein ping ?
Also, jeder Rechner in einem Netz (auch im Internet) braucht eine eigene Adresse, du hast ja auch eine eindeutige Adresse deiner Wohnung, oder? In Netzen die mit dem Protokoll TCP/IP (dazu gleich mehr) betrieben werden, ist es eben die IP-Adresse.
Mit einem ping kannst du feststellen ob ein Rechner erreichbar ist.

Im Detail sieht das Ganze dann so aus:

- Dein Rechner weiß zwar welche IP-Adresse er nehmen muß, aber nicht die MAC-Adresse. Jede Ethernet-Karte hat ihre eigene Nummer, diese bekommt sie bei der Produktion "eingebrannt". Ein Beispiel: 00:50:da:35:00:70 . Anhand dieser Adresse schickt nun deine Ethernetkarte die Pakete ab. Warum gibt es denn IP-Adressen und Mac-Adressen ?
  Tja, das geht schon fast zu weit, also kurz: Eine IP-Adresse ist eine "höherwertige Adresse", sie dient den Programmen als Identifikation und wird im Internet benutzt. Eine Mac-Adresse wird nur von den Ethernet-Karten im näheren Umkreis benutzt, also in deinem Netzwerk.
  Diese Mac-Adresse braucht dein Rechner jetzt, deswegen macht er einen "arp" Rundruf und bekommt dann gesagt welche Netzwerkarte für die IP-Adresse 192.168.15.66 "zuständig" ist.

- Dann schickt dein Rechner ein Datenpacket ab. Damit aber der andere Rechner auch weiß was das ist, wurden die Protokolle eingeführt. Ein Protokoll sagt aus, in welchem Format das Paket ist. Für ping wird das Protokoll mit dem Namen Internet Control Message Protocol, oder kurz ICMP verwendet. Nicht wundern, trotz des Namens hat es nicht direkt mit dem Internet zu tun ....

- Dein Paket geht auf die Reise durch das Kabel ...

- Dein Paket kommt an und der Rechner beantwortet es. Dazu fragt er wieder nach, welche Mac-Adresse zu dieser IP-Adresse gehört, und schickt dann ein Antwortpacket.

- Dein Rechner bekommt das Antwortpacket, wertet es aus und schreibt dir eine Meldung. Dann schickt er wieder ein Paket, und das Spiel beginnt von vorne. Fast, denn die Mac-Adresse wird eine kurz Zeit gespeichert und muss nun nicht wieder neu angefragt werden.

Übrigens, ein ping unter Linux lässt sich mit "Strg+c" abbrechen ...

WWW

Dann kommen wir mal zum nächsten Beispiel:
Du bist mit dem Internet verbunden und gibst in deinem Webbrowser www.tuxhausen.de ein.

Im Detail sieht das Ganze nun etwas anders aus:

- Zuerst mal braucht dein Rechner die IP-Adresse von "www.tuxhausen.de". Damit er diese bekommt, muss er einen DNS-Server fragen. DNS steht für Domain Name System und dient dazu den vollen Namen eines Rechner in eine IP-Adresse zu wandeln.
  Normalerweise fragt dein Rechner dazu den DNS-Server deines Providers; wenn dieser die IP-Adresse kennt, sagt er sie dir. Wenn er die Adresse nicht kennt, fragt er einen anderen DNS-Server, der wiederum fragt evtl. einen anderen ... und irgendwann hast du die IP-Adresse.
  Damit das klappt darf natürlich zu jedem Namen nur eine IP-Adresse gehören, deshalb gibt es mehrere Organisationen die Namen und IP-Adressen vergeben. In Deutschland ist dafür das Denic zuständig.
  Übrigens verwendet DNS meist das Protokoll UDP (User Datagramm Protocol).

- Den Rechner weiß nun wohin mit dem Paket, und er weiß auch, dass er das Protokoll TCP/IP nehmen soll. TCP/IP besteht eigentlich aus zwei Protokollen: TCP (Transmission Control Protocol) sorgt dafür das den Paket auch wirklich ankommt (sonst wird es erneut gesendet) und IP (Internet Protocol) sucht den Weg durch das Internet.
  Damit die Sache für den empfangenden Rechner noch einfacher wird, gibt es die Ports. Diese kann man mit einer Hausnummer vergleichen:
  Sagen wir mal deine IP-Adresse wäre eine Straße, in dieser Straße erwartet jeder Bewohner ein Paket, aber jeder ein anderes (das wären die Protokolle). Jetzt kann der Paket-Bote von Haus zu Haus wandern, bis er den findet der das Paket bekommt, oder er bekommt die Hausnummer gesagt. Dann weiß er direkt wo er hin soll und ist schneller. Ein Port ist also quasi eine Hausnummer .
  Für Web-Server wird der Port 80 benutzt.

- Dein Rechner schickt nun das Paket ab und es läuft nun quer durch das Internet. Es läuft über etliche Rechner, das interessante bei TCP/IP ist folgendes: Es kann sein das ein Paket über Asien läuft und das andere über Europa, egal wo der Server steht. TCP/IP sucht sich immer einen Weg, deshalb ist das Internet so schön ausfallsicher. Das Internet war nämlich eigentlich mal ein Militärnetz mit dem Namen Arpanet...

- Irgendwann (so nach einigen hundert langen Millisekunden) kommt das Paket beim Webserver (bei Linux-System meist das Programm Apache) an. Dieser beantwortet es und schickt es dir zurück ....

Die IP-Adresse von "www.tuxhausen.de" ist übrigens 217.160.16.47 und mit dieser könntest du auch surfen (wenn es kein virtueller Server wäre). Aber wer will sich bei so einem schönen Namen eine hässliche Nummer merken ;-))

Einrichtung einer Netzwerkkarte

Hier erst mal eine Warnung: Ich stelle hier zwar dar wie man ein Netzwerkkarte ins Netzwerk manuell einbindet. Aber es ist besser die Tools des Distributors zu benutzen, da diese die ganze Konfiguration auch prüfen können !

Module laden

Zuerst musst du deine Karte hardwaremäßig in Betrieb nehmen. Dazu kannst du den Treiber fest in den Kernel einbinden (dazu musst du aber einen neuen Kernel kompilieren) oder ein Modul laden.
Da alle Distributionen sowieso hunderte von Modulen zu dem Kernel kompiliert haben, ist es einfacher und ausreichend ein Modul zu laden.

Dazu musst du erstmal wissen was für eine Karte du hast. Dabei ist es nicht wichtig ob die Karte von "TuxTech" (oder sonstwem Billiganbieter) stammt, sondern nur welcher Chip auf der Karte ist. Bei den billigen ist es im Moment meist ein RealTek-Chip, bei Boards mit integrierter Netzwerkkarte meist ein VIA, Intel oder Marvell-Chip. Wer eine Qualitätskarte von 3Com oder Intel hat, hat eh kein Problem mit dem Treiber. Also, vor dem Einbau der Karte gucken was auf dem großen Chip steht.
Ich gehe mal davon aus, dass du hast eine TuxTech-100 PCI-Karte mit einem RealTek 8139 Chip hast.

Dann laden wir nun endlich (als root) das Modul:

```
[root]# modprobe rtl8139
```

Jetzt sollte kein Fehler auftreten, andernfalls ist der Treiber wohl doch falsch. Bei uralten ISA-Karten muß evtl. noch die IO-Adresse und der IRQ angeben werden.

Angeben einer IP-Adresse

Jetzt geben wir der Karte eine IP-Adresse, aber wir können ihr natürlich nicht irgendeine geben. Warum nicht ?
Gibst du deiner Karte die Adresse 217.160.16.47, wirst du Probleme haben diese Seite zu sehen. Denn diese IP-Adresse gehört zu "www.tuxhausen.de". Gut, wir brauchen also eine IP-Adresse, dann müssen wir eine beantragen....
Wenn dein Rechner FEST (also ohne Einwahl oder Provider) im Internet hängen würde, stimmt das schon. Aber das machst du normalerweile zuhause (und in kleinen Firmen) eh nicht. Und genau für diesen Zweck gibt es die privaten IP-Adressen, diese werden nicht im Internet verwendet und dein Provider leitet sie nicht weiter. Und das sind die privaten Adressen:

10.0.0.0	- 10.255.255.255	Klasse A Netz
172.16.0.0	- 172.31.255.255	Klasse B Netz
192.168.0.0	- 192.168.255.255	Klasse C Netz

Am einfachsten ist es eine Adresse aus dem Klasse C Netz zu nehmen, nehmen wir 192.168.1.1.
Das sagst du deiner Netzwerkkarte:

```
[root]# ifconfig eth0 192.168.1.1
```

"eth0" ist der Name der ersten Netzwerkkarte unter Linux, danach steigt einfach die Nummer, also eth1, eth2...
Schauen wir uns mal das Ergebnis an:

```
[root]# ifconfig -i eth0
```

Hier steht nun deine IP-Adresse (inet addr:192.168.1.1) und auch die MAC-Adresse deiner Netzwerkkarte (HWaddr 00:50:da:35:00:70).

Dein Rechner hat nun eine IP-Adresse, wenn du noch weitere Rechner hast, solltest du diesen nun auch IP-Adressen verpassen. Aber niemals zweimal dieselbe Adresse, das gibt nur Ärger im Netz.

Netzwerk-Routing

Jetzt haben zwar alle Rechner ihre IP-Adressen, nur wissen sie noch nicht das sie alle im selben Netz sind.
Also geben wir ihnen ein Netz, das wäre für unser Klasse C Netzwerk das Netz 192.168.1.0 mit der Netzwerkmaske 255.255.255.0.
Warum ? Das hier zu erklären geht wirklich zu weit, das brauchst du nur wenn dein Netzwerk zu langsam wird. Dann mußt du das Netz in Unternetze (Subnets) teilen und diese dann routen.
Dann sagen wir das mal unserem Linux-Rechner:

```
[root]# route add -net 192.168.1.0 netmask 255.255.255.0
```

Diese Zeile ist bei allen Rechner im Netz gleich !!! Keine Nummer ändern !!!
Den Erfolg deiner Aktion kannst du auch mit route überprüfen:

```
[root]# route -n
```

Jetzt kannst du schon mal den ersten Test machen, und zwar vom Rechner mit der IP 192.168.1.1 den Rechner 192.168.1.2 anpingen:

```
[tux]$ ping 192.168.1.2
```

Das sollte nun gehen, ansonsten nochmal alle Einstellungen genau nachschauen.

DNS

Das hat ja schon schön funktioniert, aber wäre es nicht schöner den Rechner auch mit einem richtigen Namen anzusprechen?
Jetzt kommt wieder das schon weiter oben erwähnte DNS zum Tragen, aber einen eigenen DNS-Rechner zu betreiben wäre im privaten Netz wohl etwas übertrieben.
Du braucht unter Linux nur eine Datei zu editieren und diese auf alle Rechner kopieren, und zwar die Datei `/etc/hosts`.
Aber erst mal zum Namen. Ein Name im Netzwerk besteht aus dem Rechnernamen und dem Domainnamen: bei "www.tuxhausen.de" ist "www" der Rechnername und "tuxhausen.de" die Domain (Ok, eigentlich ist "tuxhausen" die Domain und "de" die Top-Level-Domain). Der Domainname braucht also kein "de" oder so. Also nennen wir unsere Domain einfach "pinguin", die Rechner selber sollten ja schon Namen haben (nehmen wir mal fisch und eis).
Dann öffnen wir die Datei `/etc/hosts` mit einem beliebigen Editor (ja, auch mit vi ;-)) und editieren diese wie folgt:

```
#/etc/hosts

127.0.0.1       localhost                   #Loopback-Interface (stehen lassen)

192.168.1.1     fisch.pinguin     fisch     #IP-Adresse von fisch.pinguin

192.168.1.2     eis.pinguin       eis       #IP-Adresse von eis.pinguin
```

Da wir ja zu faul sind jedesmal den kompletten Namen (Name+Domainname) anzugeben, machen wir uns die Sache einfach.
Durch den Eintrag "search pinguin" in der Datei `/etc/resolv.conf` werden unvollständige Namen (eis) um die Domain erweitert (zu eis.pinguin):

```
#/etc/resolv.conf

 search pinguin              #Unvollständige Namen gehören zu .pinguin
```

Wer einen echten Nameserver für Verfügung stehen hat, trägt hier einfach `nameserver 192.168.1.111` ein, dabei hat der Nameserver natürlich die Adresse 192.168.1.111.

So, diese Dateien müssen jetzt auf jeden Rechner im Netzwerk kopiert werden !!! Und wenn du an einer Datei etwas änderst, mußt du es auch an allen anderen machen.

OK, testen wir unser Netzwerk:

```
[tux]$ ping eis
```

Sollte nun an jedem Rechner gehen, toll nicht !!
Damit steht dein Netzwerk !!

Gateway

Du siehst zwar jetzt alle deine Rechner, aber ins Internet kommt nur der Rechner mit der ISDN-Karte (oder Modem, oder ...). Dir fehlt der Gateway, dieser übernimmt alle Anfrage die nicht im lokalen Netz beantwortet werden können.
Fein raus ist jetzt derjenige der einen eigenen Router im Netzwerk hat, der die Verbindung ins Internet herstellt. Mit Router meine ich einen Hardware-Router von Firmen wie Cisco, Funkwerk BinTec, Fritz!, NetGear ...

Wenn du so einen Router hast, mußt du diesem auch eine IP-Adresse in deinem Netz verpassen (wie das geht steht im Handbuch). Deinen Linux-Rechner mußt du nun noch mitteilen welche Adresse der Router (hier 192.168.1.200) hat:

```
[root]# route add default gw 192.168.1.200
```

Alle die nur einen Linux-Rechner mit ISDN-Karte (oder so) haben, müssen aus diesem einen Gateway basteln. Leider ist das nicht so ganz einfach, du mußt IP-Forward aktivieren und die IP-Adressen aus deinem Netz maskieren. Dazu mußt du dir aber eine Firewall bauen ...
Ein Beispiel findet du auch in tuxhausen, aber guck erstmal in das Handbuch deiner Distri. Fast alle bieten dafür recht einfache Möglichkeiten an.

So, das wars. Ich beende hier die Konfiguration und gebe dir noch einige Tipps mit auf den Weg:

- Alle Dinge die du eingetippt hast, sind beim nächsten Start wieder weg. Also ein Init-Skript basteln (das kam ja auch schon im Kurs vor)

- Wenn du mehr über Netzwerkeln wissen willst, hol dir ein gutes Buch.
  Zum Beispiel Linux - Wegweiser für Netzwerker (zweite Ausgabe) (das gibt es online) oder viele `man`-Pages lesen ;-))
  Wenn du IP-Adressen so richtig verstehen willst, solltest du dir Understanding IP von 3Com antun.

- Wenn du Rechner fernadministrieren (also nicht vor dem Rechner sitzt) willst, geht das mit Linux wunderbar. Am besten nimmst du dazu OpenSSH

Einige nützlich Netzwerk-Tools

Hier gebe ich mal einen kleinen Überblick welche Tools im Netzwerk sinnvoll sind und jeweils ein Beispiel wie du sie benutzt.

ping	Erreichbarkeit des Rechnern überprüfen `[tux]$ ping eis.pinguin`	
route	Zeigt die Netzwerk-Routing-Tabelle an `[root]# route -n`	
arp	Zeigt die letzten MAC-Adressen an `[root]# arp -a`	
tcpdump	Liest ALLE Packete im Netzwerk mit `[root]# tcpdump -i eth0`	
Wireshark	Liest ALLE Packete im Netzwerk mit, graphische Oberfläche	

A1 - Eine kleine Aufgabe

1. Lege zwei neue Benutzer an: user1 und user 2
   Zusatz: user2: Der Midnight Commander soll nach dem Einloggen automatisch geöffnet werden
2. Nun als user1 einloggen und die Verzeichnisse Daten, src und source erstellen
3. Mit welchem Parameter listet `ls` die versteckten Dateien?
4. Ins Verzeichnis src und source jeweils 5 beliebige Dateien aus verschiedenen Verzeichnissen kopieren
5. Das Verzeichnis source wieder KOMPLETT löschen
6. Den Editor GNU-nano aus dem Internet downloaden und installieren, wie ist egal (Tip: www.freshmeat.net)
7. Eine Diskette mit ext2 formatieren
8. Einen Text mit dem Editor GNU-nano schreiben (egal was, ca. 5 Zeilen) und auf die eben formatierte Diskette schreiben (Tip: nano ist ein freier pico-Clone)

9. Zusatz: Ein Skript schreiben dass beim Erreichen von Runlevel 2 einmal piept und nichts ausgibt

Einige Fragen

1. Frage: Wo stehen unter Linux die meisten Konfigurationsdateien?
2. Frage: Mit welcher Tastenkombination wechselst du von X auf Konsole 3?
3. Frage: Was macht das Kommando `script`?
4. Frage: Wie heißt der wohl älteste und urigste Unix-Editor ?
5. Frage: Wozu dient die Datei `/etc/fstab` ?
6. Frage: Was bewirkt "nano -v datei" ?
7. Frage: Was bewirkt die Taste TAB in der bash?
8. Frage: Wer oder was ist Tux ?

So das wars, der Kurs ist vorbei, du hast nun (hoffe ich) alle wichtigen Grundalgen um ein guter Tux zu werden ;-))

A2 - Lösungen

Zu der Aufgabe gebe ich bewusst keine Lösung ab, bitte versuche einfach die Aufgabe abzuarbeiten. Wenn du alles erreicht hast, hast du die Aufgabe ja geschafft und dir selber geholfen, das ist das Ziel !!

Einige Fragen und die kurzen Antworten

1. Frage: Wo stehen unter Linux die meisten Konfigurationsdateien?
   - Unter /etc/
2. Frage: Mit welcher Tastenkombination wechselst du von X auf Konsole 3?
   - Strg+Alt+F3
3. Frage: Was macht das Kommando `script`?
   - `script` schreibt alle deine Eingaben (und Ausgaben) in einer Datei mit
4. Frage: Wie heißt der wohl älteste und urigste Unix-Editor ?
   - vi
5. Frage: Wozu dient die Datei `/etc/fstab` ?
   - bestimmt die einzubindenden Dateisysteme
6. Frage: Was bewirkt "nano -v datei" ?
   - zeigt die Datei im Read-Only-Modus an
7. Frage: Was bewirkt die Taste TAB in der bash?
   - Befehl, Datei oder Verzeichnis wird vervollständigt
8. Frage: Wer oder was ist Tux ?
   - Tux ist der Pinguin und DAS Symbol/Maskottchen für Linux

B - Kernel

Der Kernel ist das Herz von Linux und kümmert sich beispielsweise um die Hardware.
Normalerweise braucht der Anwender ihn nicht zu ändern, aber manchmal doch.
Und wie das geht erläutert der folgende Teil des Linux-Kurses.

Der Kernel ist das Herz eines Betriebssystems, er regelt den Zugriff auf die Hardware, verwaltet den Speicher und die Prozesse, kurz ohne ihn geht gar nichts. Also warum zum Pinguin solltest du ihn ändern ?
Der wohl häufigste Grund ist der: Du hast die brandneue HyperDuperNet15, aber dein oller Kernel 2.6.2 kennt diese Karte nicht und du hast kein Netzwerk. Aber der neue Kernel 2.6.18 kennt diese Karte und soll damit ordentlich laufen.
Es könnte aber auch sein das du einem extrem sicheren und kleinen Kernel haben willst,
oder du willst mal richtig cool sein und einen eigenen Kernel bauen.

Wichtig ist nur folgendes: Lösche dir niemals den Kernel der gerade stabil arbeitet !!

Achtung !!!!
Mittlerweile ist es fast nicht mehr nötig einen Kernel zu kompilieren, erschwerend kommt noch hinzu das fast jede Distributionen einen anderen Weg geht.
Ich beschreibe hier den "klassischen" Weg einen Kernel zu kompilieren, bitte schaue vorher auf jeden Fall in deine Dokumentation (z.B. für Debian im der Debian-Referenz Kapitel 7)

Kernel saugen

Der erste Schritt ist es natürlich den Kernel herunterzuladen. Dazu solltest du immer den aktuellen Kernel nehmen, aber warte nach dem Erscheinen eines neuen Kernels erstmal eine Woche (außer du möchtest den "Neuen" mal richtig testen), denn manchmal gibt es auch im Linux-Kernel Fehler.
Aber was ist der aktuelle ??
Kein Problem, dafür gibt es kernel.org [1], dort findest du alle nötigen Informationen.
Dann lädst du den Kernel runter und legst du Kernel nach `/usr/src`.

Nun wird der Kernel entpackt, wenn du ihn tar.gz-File heruntergeladen hast:

```
[root]/usr/src# tar -xvzf linux-2.4.18.tar.gz
```

wenn du ihn tar.bz2-File heruntergeladen hast:

```
[root]/ausr/src# bunzip2 linux-2.4.18.tar.bz2 && tar -xvzf linux-2.4.18.tar
```

Damit liegt der Kernel nun unter `/ausr/src/linux`, das ist aber nicht so schön. Besser ist den Kernel in `/usr/src/linux-2.4.18` zu legen. Leider suchen einige Patche/Programme die Kernelquellen aber nur unter `/usr/src/linux/`; also müssen wir einen Link anlegen der von `/usr/src/linux` nach `/usr/src/linux-2.4.18` zeigt:

```
[root]/usr/src# mv linux linux-2.4.18     #linux nach linux-2.4.18 verschieben
[root]/usr/src# ln -s linux-2.4.18 linux  #link nach linux-2.4.18 anlegen
```

Patch

Neue Dinge oder Treiber finden meist nicht direkt ihren Weg in den Kernel, sondern liegen meist am Anfang nur als Patch vor. Wenn diese Patche in Ordnung sind und stabil laufen, läßt Linus sie in seinen Kernel.
Und was macht man nun mit so einem Patch ?
Zuerst mal ist es ganz wichtig das der Patch genau zu dem Kernel paßt, danach ist es Pflicht die Doku zum Patch (mindestens INSTALL) zu lesen.
Wenn du keinen Patch installieren willst, kannst du zum nächsten Absatz springen.

Als kleines Beispiel nehme ich mal den XFS Patch:

```
[root]/usr/src# patch -b -p0 < xfs-2.4.18-all-i386
```

Die Option -b sichert die original Dateien als Backup, überschreibt also nicht die Originaldatei, sondern verschiebt sie.

Kernel konfigurieren

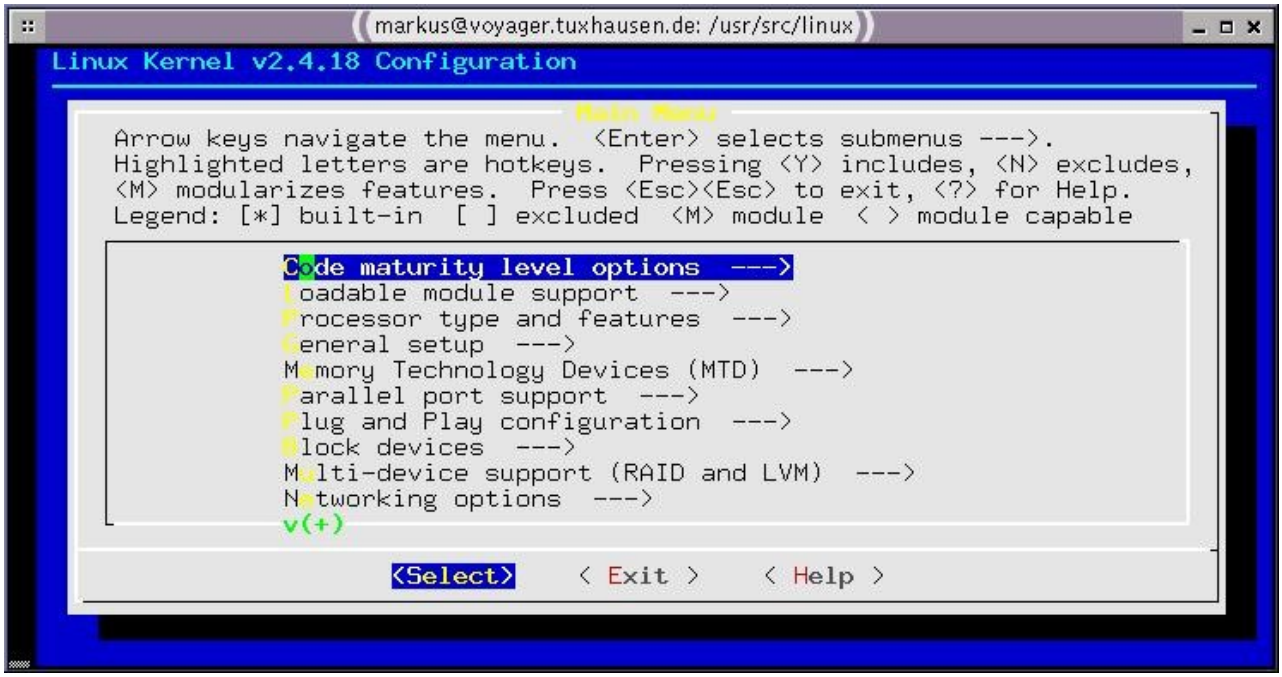
Nun kommen wir zum interessantesten Teil: Den Kernel konfigurieren. Hierbei legst du fest wie dein Kernel sein soll und was er kann (und wie groß er ist).
Dazu gibt drei verschiedene Wege, aber zuerst geht du ins Verzeichnis `/usr/src/linux` :

```
[root]/usr/src/linux# make config
```

1. Danach werden dir Unmengen von Fragen gestellt, und danach dein Kernel erstellt. Diese Methode ist die schlechteste und nur im absoluten Notfall zu gebrauchen, also schnell weiter.

```
[root]/usr/src/linux# make menuconfig
```

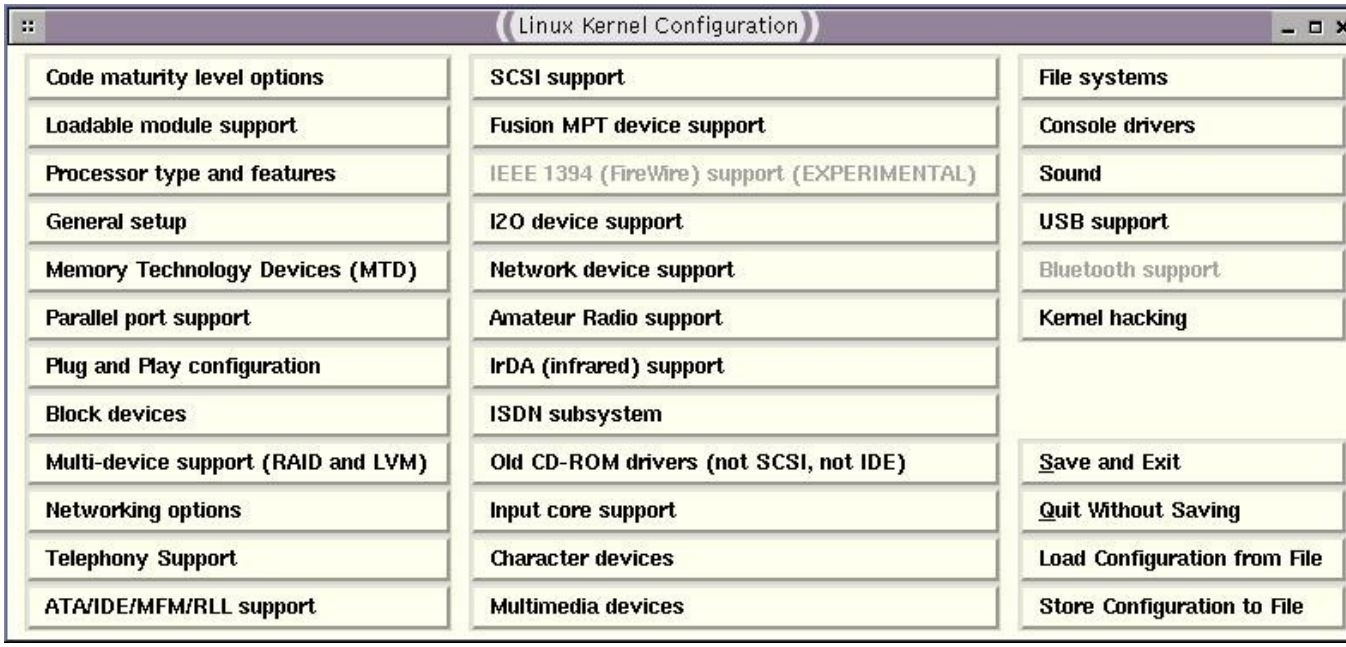
2. Hierbei erscheint in der Text-Konsole ein einfaches grafisches Menü mit dem du alles einstellst. Das Ganze sieht so aus:




```
[root]/usr/src/linux# make xconfig
```

3. Nun erscheint unter X ein nettes grafisches Fenster und du kannst dich durchklicken.
   Diese Sache ist die übersichtlichste, braucht aber ein laufendes X-System (dieses muß nicht auf dem lokalen Rechner laufen siehe Minitipps: X-Display auf anderen Rechner umleiten).

Und so sieht es unter X aus:



Bevor wir nun loslegen erst noch ein kleiner Tipp:
Wenn du deinem Kernel eine eigenen Namen geben willst (was sehr zum empfehlen ist, besonders wenn du spezielle Patches einbindest) mußt du nur eine Zeile in der Datei `Makefile` ändern:

```
EXTRAVERSION = mu-1
```

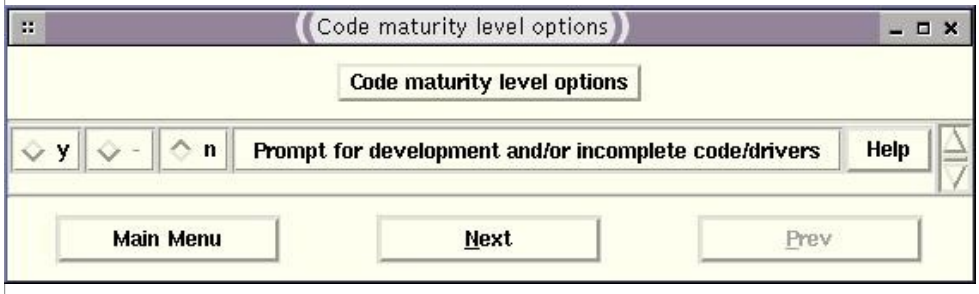
Damit taufst du deinen Kernel auf den Namen 2.4.18mu-1 .

Und nun geht es endlich mal richtig los:
Konfigurieren wir den Kernel. Ich empfehle dir `make xconfig`.
Außerdem ist es hilfreich zu wissen welche Komponenten in deinem Rechner stecken, im Zweifelsfall Karte ausbauen und auf den Chip gucken (die Nichttechniker finden am besten heraus wie ihre Karte genau heißt: Rechnung, Anleitung, Meldung beim Rechnerstart).

Damit du nicht ganz verloren im Dschungel der Kernel-Parameter/Funktionen stehst, gehen wir nun die wichtigsten Parameter zusammen durch.
Wenn du einmal weißt wie es geht, kommt du sicher mit den anderen Parameter klar. Jede Box stellt ein Menü dar, naja probiers einfach aus:

Code maturity level options --->

Nun solltest du in etwa folgendes sehen:



Zur Erklärung:
Mit "**y**" fügst du diese Funktion dem Kernel fest zu, direkt nach dem Booten hat der Kernel diese Funktion.

In der nächsten Spalte steht hier ein "-", wenn dort aber ein "**m**" steht (kommt gleich noch), fügst du diese Funk
als Modul hinzu. Das bedeutet diese Funktion ist nicht fest im Kernel eingebaut, sondern wird bei Bedarf vom
Kernel automatisch geladen.
Achtung !!
Alle Dinge die du beim Booten deines Systems brauchst, dürfen KEIN Modul sein !!
Wie soll der Kernel das Modul zum Betrieb des SCSI-Adapters von der SCSI-Platte laden wenn er keinen Treil
dafür hat (Henne-Ei-Problem) ??

Mit "**n**" entfernst du logischerweise diese Funktion aus deinem Kernel.

Der Button "**Help**" enthält eine englische Hilfe zu der Funktion. Dort steht auch meist eine Empfehlung ob du d
brauchst oder nicht.

In diesem Menü siehst du den Parameter **Prompt for development and/or incomplete code/drivers**
Wenn du diesen aktivierst (also "y") werden in den nächsten Menüs auch die Parameter/Funktionen angezeigt
die Kernel-Entwickler als experimentell oder noch nicht stabil genug ansehen. Am besten deaktivierst du diese
Punkt, wenn du die gesuchte Funktion nicht findest, aktivierst du den Punkt und suchst noch mal. Unter xconfig
sind diese Funktionen/Pararmeter heller dargestellt.
Ich gehe hier aber mal davon aus das der Button nicht aktiviert ist.

Ein Klick auf "Main Menu" und zum nächsten Punkt:

Processor type and features --->
Hier stellst du deinen Prozessor ein

Processor family
Den richtigen Prozessor eintragen, sonst läufst dein Kernel nicht. Im Zweifel auf Intel/AMD Systemen den 3
nehmen.

Symmetric multi-processing support
Wenn du nicht gerade mehrere Prozessoren im Rechner hast -> "n"

Parallel port support --->
Parallel port support
PC-style hardware
Wenn du einen Drucker oder etwas anderes am Parallel Port hängen hast -> "y" (am besten immer)

Networking options --->
Network packet filtering (replaces ipchains)
Wenn du dir eine Firewall bauen willst mußt du mindestens diesen Punkt aktivieren

ATA/IDE/MFM/RLL support --->
IDE, ATA and ATAPI Block devices --->
SCSI emulation support
Für ATAPI-Brenner brauchst du den SCSI-Emulation Support (`cdburn` braucht SCSI-Brenner)

Generic PCI IDE chipset support PROMISE PDC202{46|62|65|67|68} support
Wenn du einen der beliebten Promise IDE-Controller hast, solltest du hier "y" angeben

SCSI support --->

SCSI low-level drivers --->
Wenn du einen SCSI-Controller hast, solltest du hier den Richtigen auswählen. Hast du keinen, deaktiviere
SCSI-Zeug.

Network device support --->
Ethernet (10 or 100Mbit) --->
Wenn du eine Netzwerkkarte hast, wähle hier deine aus. Ansonsten lass es weg.

ISDN subsystem --->
ISDN support
Passive ISDN cards --->
Active ISDN cards --->
Nach den SCSI und Netzwerkkarten kommen nun die ISDN-Controller ....

Input core support --->
Hinter diesem Punkt ist die USB-Tastatur und Mouse versteckt. Am besten immer mit reinnehmen, macht d
keine Probleme mit USB.

Character devices --->
Parallel printer support
Hier steht der zweite Teil für den Parallelportdrucker.

File systems --->
DOS FAT fs support
MSDOS fs support
VFAT (Windows-95) fs support
Microsoft Joliet CDROM extensions
NTFS file system support (read only)
Diese Punkte solltest du alle aktivieren wenn du auf Windows-Dateisysteme oder CDs zugreifen willst

Network File Systems --->
SMB file system support (to mount Windows shares etc.)
Wenn du Windows-Freigaben mounten willst, aktiviere diesen Punkt.

NCP file system support (to mount NetWare volumes)
Wer noch einen Novellserver hat, braucht das NCP. Der Rest nicht :-)

Native Language Support --->
Codepage 437 (United States, Canada) (NEW)
Codepage 850 (Europe) (NEW)
Diese Funktionen können dir Ärger mit DOS/Windows Dateiname ersparen (Umlaute,....)

Console drivers --->
Frame-buffer support --->
Mit Hilfe des Framebuffers kannst du höhere Auflösungen in der Konsole erreichen (und in der linken ober
Ecke erscheint ein Tux).
Wenn deine Grafikkarte hier bei steht, nimm sie; ansonsten die VESA VGA graphics console (Nur sichtbar
"Prompt for development and/or incomplete code/drivers")

Sound --->
Ist wohl klar oder? Wenn deine Karte dabei ist, anklicken

USB support --->
Hier kommen die USB-Geräte an die Reihe, am besten alles notwendige als Modul.

So das waren die wichtigsten Parameter/Funktionen, damit sollte dein Kernel schon mal alles Wichtige unterstützen. Spezielle Treiber oder Dinge lassen sicht meist nur durch suchen finden.

Gut, nun solltest du deine Konfiguration erst noch unter einem anderen Namen speichern (Save Configuration to file), so was in der Art "kernelconfig_2.4.18-mu-1" nehme ich immer. Die Nummer erhöhst du einfach mit jedem Kernel-Versuch.

So nun verlassen wir das Menü (Save und Exit, ja wir wollen speichern) und bauen den Kernel.

Kernel bauen (oder besser kompilieren)

Das ist ganz einfach, du gibst folgendes ein:
```
[root]/usr/src/linux# make dep && make clean && make bzImage
```
Damit wird der neue Kernel gebaut, das kann je nach Rechenleistung ein gute Weile dauern .....
Jetzt fehlen noch die Module, also bauen wir die auch noch (das geht schneller):
```
[root]/usr/src/linux# make modules && make modules_install
```
Die Module findest du nach der Installation unter /lib/modules/2.4.18mu-1 .
Gut die Module sind da, aber wo ist der Kernel ??
Auf einem Intel/AMD-System findest du ihn unter /usr/src/linux/arch/i386/boot/bzImage. Da wir ihn aber dort mit dem Namen nicht brauchen können, wird er umkopiert:
```
[root]/usr/src/linux# cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-2.4.18mu-1
```

Jetzt kommt noch eine kleine Hürde:
Dein Bootmanager muß wissen wo der neue Kernel liegt. Nochmal zur Erinnerung: Den alten Kernel bloß nicht löschen !!

Wenn du den **lilo** verwendest, trägst du in deine `lilo.conf` (meist `/etc/lilo.conf`) folgendes zusätzlich ein:
```
image=/boot/vmlinuz-2.4.18mu-1
label=Neuer Kernel 2.4.18mu-1
root=/dev/hda5
```

Das kannst du so nicht übernehmen, sondern muß angepasst werden. Siehe die einfach die anderen (alten) Einträge mal an.

Jetzt mußt du umbedingt noch den `lilo` aufrufen:

```
[root]/usr/src/linux# lilo
```

Benutzt du den grub, so trage folgendes in deine `menu.lst` (meist `/boot/grub/menu.lst`) ein:

```
title Neuer Kernel (2.4.18mu-1)
kernel (hd0,4)/boot/vmlinuz-2.4.18mu-1 root=/dev/hda5
```

Neustart

Jetzt wird es wirklich spannend. Starte deinen Rechner neu und boote deinen Kernel.....

Ich hoffe mal es hat geklappt, wenn nicht, stimmt wahrscheinlich der Prozessor-Typ oder so was in der Art nicht.
Kein Problem starte mit dem alten Kernel und konfiguriere/baue den Kernel neu.

Damit auch siehst das dein Kernel wirklich der Laufende ist, gebe man das hier ein:

```
[root]/# uname -rs
```

Super, dein Kernel läuft !!!
Wahrscheinlich wirst du bald merken, das was fehlt (passiert mir immer), also noch mal konfigurieren und bauen :-))