Eine Shell (engl. Hülle, Schale, Außenhaut) bezeichnet die traditionelle Benutzerschnittstelle unter Unix-Betriebssystemen. Der Benutzer kann in einer Eingabezeile Kommandos eintippen, die der Computer dann sogleich ausführt. Man spricht darum auch von einem Kommandozeileninterpreter.

Die Bash (Bourne-again-shell - eine Weiterentwicklung der Bourne-Shell) ist die Standard-Shell auf den meisten Linux-Systemen und wurde auf fast alle Unix-Systeme portiert.
Bash? Wozu brauche ich das?!

Unter Windows spielen Kommandozeileninterpreter für den Durchschnittsbenutzer heute eine geringe Rolle. Für scheinbar alle Zwecke gibt es graphische Tools, die sämtliche Konfigurations- und Bedienungsschritte erledigen können. Scheinbar - denn auch unter Windows kommen professionelle Systemadministratoren nicht ohne Kommandozeile aus.

Unter Linux gibt es eine ähnliche Tendenz: So manche moderne Distribution versteckt die Shell heute ebensogut wie der große Konkurent aus Redmont. Dennoch ist sie ein wichtiges Instrument, mit dem sich jede/r vertraut machen sollte.

Es gibt viele nützliche Programme, für die es keine graphische Oberfläche gibt und auch Programme mit graphischer Oberfläche kennen oft Optionen, die nur von der Kommandozeile aus zugänglich sind.
Im Störungsfall ist eine Rettungs-Shell oft das einzige zur Verfügung stehende Mittel, um das Problem zu beheben, aber auch wenn das System nicht streikt, kommt es gelegentlich vor, dass ein Programm nach einem Update nicht mehr funktioniert. Unter Windows bleibt da meist nur warten und hoffen. Unter Linux starte ich das widerspenstige Programm einmal als Shell-Kommando und sehe mir an, was es zu sagen hat. Die Fehlermeldungen sind häufig schon aussagekräftig, aber wenn nicht, gebe ich sie in eine Suchmaschine ein und erfahre so nicht nur, warum das Programm nicht läuft, sondern auch meist, wie ich es zum Laufen bringen kann.

Was mich an Windows immer genervt hat, war, dass es Probleme gab, bei denen ich an einen Punkt kam, wo ich aufgeben musste. Unter Linux gibt es auch Probleme, aber ich weiß, dass ich, eventuell mit einigem Aufwand, die Lösung finden werde.

Noch einige Tipps:

Moderne Terminal-Emulationen beherrschen meist copy/paste. Das heißt, ich kann mit der linken Maustaste einen Teil der Ausgabe (zum Beispiel Dateinamen) markieren, mit der rechten Maustaste kopieren und wieder mit rechter Maustaste in der Befehlszeile einfügen. Auch aus anderen Dokumenten kopierter Text kann so eingefügt werden.

Die Bash speichert alle eingegebenen Befehle. Mit den Pfleiltasten ↑ und ↓ können sie zurückgerufen und (eventuell modifiziert) wieder abgesetzt werden. Sollen alle gespeicherten Befehle ausgegeben werden, genügt die Eingabe history, gefolgt von ENTER.

Tippt man den Anfang eines Befehls und drückt dann die Tabulatortaste, versucht die Bash den Befehl zu vervollständigen.

Dies geschieht soweit der Befehl eindeutig ergänzt werden kann. Gibt es mehrere Möglichkeiten, so wird nur bis zu dem Punkt vervollständigt, ab dem sich die Möglichkeiten unterscheiden. Drückt man dann 2x die Tabulatortaste, werden die Möglichkeiten aufgelistet. Tippt man dann weiter und drückt abermals die Tabulatortaste, wird auch der Rest ergänzt.

---

(1) Das System erkunden

Zunächst öffnen wir ein Terminal-Fenster. Wenn im Menü nicht zu finden, können wir die Suchfuntion aufrufen und "terminal" suchen (die ersten drei Buchstaben einzutippen wird reichen).

Betrachten, Navigieren, Suchen - pwd, ls, cd und find

Nach dem Aufruf der Shell befinden wir uns in unserem Home-Verzeichnis. Am Anfang der Zeile, in die wir schreiben können, steht etwas wie benutzer@computer:~$ - vor dem $ können noch Pfadangaben zu finden sein, aber wenn dort nur eine Tilde steht, befinden wir uns im Home-Verzeichnis.

Um das zu überprüfen können wir uns mit pwd (print work-directory) den Pfad des aktuellen Verzeichnisses anzeigen lassen:

pwd

Wir sehen uns das "aktuelle" Verzeichnis an, das heißt, das Verzeichnis, in dem wir uns gerade befinden:

ls

Wir bekommen alle Dateien und Verzeichnisse im Home-Verzeichnis aufgelistet.
Die gleiche Ausgabe in detailierter Form erhalten wir mit:

ls -l

Das -l ist eine Option. Es steht hier für eine "lange" Ausgabe - zu jeder Datei bekommen wir Informationen über Größe und Dateirechte (zu diesen kommen wir etwas später).
Allgemein gibt ls (list) den Inhalt des angegebenen Verzeichnisses aus:

ls (-option) verzeichnisname

Wenn wir zuvor nur ls ohne Verzeichnisname ausgeführt haben, so geht das nur, weil dies als ls . interpretiert wird, wobei der Punkt immer für das aktuelle Verzeichnis steht.
Im Home-Verzeichnis gibt es das Verzeichnis Desktop. Darin befinden sich die Dateien, die wir auf unserem Desktop sehen.
Verzeichnis "Desktop" ausgeben:

ls Desktop

Wenn wir ein frisch installiertes System haben, wird da unter Umständen noch nicht viel zu sehen sein. Wenn wir gar nichts auf dem Desktop liegen haben, wird die Ausgabe sogar leer sein.
Wir können dann schauen, ob überhaupt etwas in einem der Verzeichnisse liegt.
Den Inhalt aller Verzeichnisse ausgeben:

*ls **

Ein Stern steht für alle Datei- oder Verzeichnisnamen und so sehen wir auch den Inhalt aller Verzeichnisse - ls gibt, wie gesagt, den Inhalt des angegeben Verzeichnisses aus und Stern bedeutet "alle"!
Wobei "alle" nicht ganz korrekt ist. Dateien und Verzeichnisse, die mit einem Punkt beginnen, werden unter Linux als "versteckt" betrachtet und normalerweise nicht angezeigt. Wollen wir auch versteckte Dateien anzeigen, müssen wir wieder eine Option benutzen.
Auch versteckte Dateien ausgeben:

ls -a

Jetzt haben wir wohl eine etwas umfangreichere Ausgabe! Um den Anfang zu sehen, müssen wir ein Stück hinaufscrollen. Dort sehen wir zwei seltsame Verzeichnisse: . und .. - diese bezeicnen das aktuelle Verzeichnis und das diesem übergeordnete Verzeichnis. Die beiden gibt es in jedem Verzeichnis! Über deren Bedeutung werden wir in Kürze mehr erfahren. Da es sie aber ohnehin überall gibt, wollen wir sie im Allgemeinen nicht anzeigen. Mit Option -A bekommen wir auch wieder versteckte Dateien ausgegeben, aber ohne . Und ...
Auch versteckte Dateien ohne . und .. ausgeben:

ls -A

Optionen können wir auch kombinieren. Nach dem Minus müssen sie nur aneinandergereiht werden:

ls -lA

liefert versteckte und nicht versteckte Dateien in detailierter Form (mit großem "A" ohne die Verzeichnisse . und ..).

Nun wollen wir aber das Home-Verzeichnis auch verlassen. Mit cd (change directory) können wir uns zwischen den Verzeichnissen bewegen.

cd verzeichnisname

Wir wechseln ins Verzeichnis "Desktop":

cd Desktop

Mit ls -a können wir die Dateien dieses Verzeichnisses anzeigen lassen. Wenn es sonst keine Dateien gibt, so sehen wir jedenfalls wieder . und ... Jetzt benützen wir .. um wieder zurück ins (übergeordnete) Home-Verzeichnis zu kommen.
Wechsel ins übergeordnete Verzeichnis:

cd ..

Wenn wir cd .. wiederholen erreichen wir das Verzeichnis, in dem als Unterverzeichnis auch unser Home-Verzeichnis liegt.

ls zeigt uns das. Wir befinden uns jetzt im Verzeichnis /home. Würden wir cd .. nochmal wiederholen, wären wir im Wurzelverzeichnis, aber dort kommen wir auch anders hin.
Wechsel ins Wurzelverzeichnis:

cd /

Wollen wir ins zuvor benutzte Verzeichnis zurück gibt es eine spezielle Option.
Wechsel ins zuletzt benützte Verzeichnis:

cd -

Um sofort ins Home-Verzeichnis zu gelangen, geben wir cd ohne etwas danach ein.
Wechsel ins Home-Verzeichnis:

cd

(cd ohne Verzeichnis steht für cd ~, wobei ~ das Home-Verzeichnis des jeweiligen Benutzers ist)

Wenn wir wissen, wo's hingehen soll, können wir auch direkt ein Verzeichnis angeben:

cd /usr/share/pixmaps

Nun sind wir in dem Verzeichnis, in dem die Bilddateien für die Desktopicons liegen.
Will ich direkt in das Unterverzeichnis "Desktop" springen, kann ich die Tilde für das Home-Verzeichnis nutzen:

cd ~/Desktop

(Ausgeschrieben wäre das cd /home/benutzername/Desktop)

Unter Linux werden alle Festplatten, Partitionen und andere Geräte wie CD-Laufwerk und USB-Stick in einen Verzeichnisbaum "eingehängt". Das grundlegende Verzeichnis ist die Wurzel /. Nun können wir im ganzen Verzeichnisbaum herumflitzen. Wir können dabei nichts ernsthaft kaputt machen, denn dazu haben wir als normaler Benutzer gar nicht die nötigen Rechte.

Üben:

cd /
ls
cd var
pwd

usw.

Wir können auch nach Dateien suchen:

find verzeichnisname -name dateiname

Wir übergeben find dazu den Verzeichnisnamen, wo wir zu suchen beginnen wollen und den Namen der Datei - wir wählen "syslog", da dieses auf jedem System vorhanden sein sollte:

find /var -name syslog

Wahrscheinlich erhalten wir außer dem Aufenthaltsort der Datei noch einige Fehlermeldungen von Unterverzeichnissen, für die wir keine Leseberechtigung haben. Im Augenblick müssen wir das so hinnehmen.

Textdateien lesen - less

Jetzt wissen wir, wo "syslog" zu finden ist, aber vielleicht möchten wir auch sehen, was darin geschrieben steht. Dabei hilft uns der Textbetrachter less:

less dateiname

Um syslog zu lesen geben wir ein:

less /var/log/syslog

(um den Pfad nicht abtippen zu müssen, können wir ihn auch aus der vorherigen Ausgabe von find kopieren - siehe auch "Tipps" ganz oben)

In "less" kann man mit den Pfeiltasten ↑, ↓, Bild↑ und Bild↓ navigieren, mit [Ende] ans Ende und [Pos1] an den Anfang springen. Beenden lässt sich "less" mit der Taste [Q].

---

(2) Dateimanipulationen

Dateien anlegen und kopieren - touch, mkdir und cp

Mit touch können wir leere Dateien erzeugen:

touch dateiname

Zunächst erzeugen wir die Testdatei "datei.txt" im tmp-Verzeichnis:

touch /tmp/datei.txt

Jetzt wechseln wir mit cd ins Home-Verzeichnis.

cd

Zum Anlegen eines Verzeichnisses benutzen wir mkdir (make directory):

mkdir verzeichnisname

Wir legen ein Testverzeichnis "mond" an:

mkdir mond

Mit cp (copy) können wir Dateien kopieren.

cp quelle ziel(verzeichnis)

Jetzt wollen wir unsere Testdatei aus /tmp in unser Testverzeichnis kopieren.

cp /tmp/datei.txt mond

Um das Verzeichnis "mond" ohne weitere Pfadangabe ansprechen zu können, müssen wir uns im richtigen Verzeichnis (im Beispiel das Home-Verzeichnis) befinden. Wenn das aktuelle Verzeichnis (zu überprüfen mit pwd) ein anderes im Verzeichnisbaum ist, müssen wir auch angeben, wo "mond" zu finden ist.
In unserem Beispiel also: cp /tmp/datei.txt ~/mond

Wir können der Datei aber auch einen neuen Namen geben. "datei.txt" wird als "kopie.txt" ins Verzeichnis mond kopiert.

cp /tmp/datei.txt mond/kopie.txt

Ein ls mond überzeugt uns vom Erfolg.

Wir haben jetzt schon gesehen, dass es zwei verschiedene Arten von Pfadangaben gibt:
mond/kopie.txt ist ein relativer Pfad. Er enthält die Unterverzeichnisse ausgehend von unserem aktuellen Verzeichnis.
/tmp/datei.txt ist ein absoluter Pfad. Er beginnt mit / für das Wurzelverzeichnis und enthält alle Unterverzeichnisse ab diesem. Er ist von überall im Dateibaum ausgehend eindeutig!

An sich überschreibt cp Dateien ohne Rückfrage. Soll vor dem Überschreiben eine Rückfrage erfolgen, kann man die Option -i benutzen:

cp -i /tmp/datei.txt mond/kopie.txt

Die Frage, ob wir die vorhandene Datei überschreiben wollen, können wir mit j(a) oder n(ein) beantworten.
Will man ein ganzes Verzeichnis kopieren, kann man die Option -R benutzen. Zunächst legen wir mit mkdir noch ein weiteres Verzeichnis an:

mkdir Mond

Wir sehen: unter Linux werden Groß- und Kleinschreibung unterschieden. "Mond" und "mond" sind verschiedene Dateien. Jetzt kopieren wir das Verzeichnis "mond" mitsamt Inhalt in "Mond":

cp -R mond Mond

Wir erhalten ein Verzeichnis "mond" im Verzeichnis "Mond". Will man nur den Inhalt und nicht auch das Verzeichnis kopieren, kann man als Quelle mond/* festlegen – die "Wildcard" * bedeutet wieder "alle Dateien im Quellverzeichnis":

cp -R mond/ Mond*

(eigentlich ist -R hier nicht notwendig, da es im Quellverzeichnis "mond" keine Unterverzeichnisse gibt)

Jetzt haben wir die Dateien "datei.txt" und "kopie.txt" auch direkt in "Mond".
Beim Kopieren können auch die beiden Verzeichnisse . und .. verwendet werden. Wechseln wir zunächst ins Verzeichnis "Mond/mond":

cd Mond/mond

Wir kopieren die Datei "kopie.txt" unter dem Namen "neu.txt" ins übergeordnete Verzeichnis:

cp kopie.txt ../neu.txt

Nun die Datei "neu.txt" zurück ins aktuelle Verzeichnis:

cp ../neu.txt .

Jetzt kopieren wir alle Dateien aus dem aktuellen Verzeichnis ins übergeordnete Verzeichnis. Weil dort schon die gleichnamigen Dateien liegen, wollen wir das Überschreiben mit j bestätigen:

*cp -i * ..*

(die Wildcard * steht wieder für alle Dateien im aktuellen Verzeichnis)
Verschieben und umbenennen - mv

Zum Verschieben, wie auch zum Umbenennen benutzen wir mv (move):

mv quelle ziel(verzeichnis)

Wir wechseln zunächst mit cd wieder ins Home-Verzeichnis. Anschließend verschieben wir "neu.txt" vom Verzeichnis "Mond" nach "mond":

mv Mond/neu.txt mond

Jetzt finden wir "neu.txt" nicht mehr im Verzeichnis "Mond", sondern nur noch in "mond".
Wir wollen aber nicht, dass sie dort "neu.txt" heißt, sondern "alt.txt":

mv mond/neu.txt mond/alt.txt

So heißt sie nun!
Weil das mit "Mond" und "mond" zu verwirrend ist, wollen wir "mond" überhaupt umbenennen. "mond" soll "anders" heißen:

mv mond anders

Ein ls bestätigt uns - kein "mond" mehr, dafür "anders".
Dateien löschen - rm und rmdir

Zum Löschen von Dateien benutzen wir rm (remove):

rm (-optionen) dateiname

Achtung:rm mit Administratorrechten ausgeführt kann im schlimmsten Fall das gesamte Betriebssystem löschen. Und auch als normaler Benutzer kann ich immer noch das Home-Verzeichnis und alle darin befindlichen Dateien vernichten.

Wir wechseln zunächst ins Verzeichnis "Mond/mond" und erzeugen dort noch einige Dateien:

cd Mond/mond
touch datei1.txt datei2.txt datei3.log

ls um zu sehen, welche Dateien nun vorhanden sind. Eine einzelne Datei löschen wir relativ gefahrlos mit:

rm datei1.txt

Man kann aber auch hier wieder * benutzen, um mehrere Dateien auf einmal zu eliminieren:

*rm *.txt*

löscht alle Datein in "mond", die auf ".txt" enden. Wir überprüfen das mit ls.

Noch gefährlicher ist es, alle Dateien im aktuellen Verzeichnis zu löschen. Wir sollten umbedingt zuvor überprüfen, in welchem Verzeichnis wir uns gerade befinden (pwd).

*rm **

Jetzt befinden sich keine Dateien mehr im aktuellen Verzeichnis. Wir wechseln ins Verzeichnis "Mond":

cd ~/Mond

Am gefährlichsten ist rm, wenn wir die Option -R verwenden. Normalerweise kann rm nur normale Dateien und Links löschen. Der Versuch ein Verzeichnis zu löschen (rm mond) wird mit einer Fehlermeldung quittiert.

*rm -R **

löscht auch alle Unterverzeichnisse mit allen darin enthaltenen Dateien!
Also, immer vor dem ENTER innehalten und überlegen: Will ich das wirklich alles löschen?!

Zum Löschen von leeren Verzeichnissen gibt es auch rmdir (remove directory):

rmdir verzeichnisname

Dieser Befehl ist weniger effektiv als rm -R, aber auch erheblich weniger gefährlich.
Nachdem wir mit cd ins Home-Verzeichnis gewechselt haben, löschen wir das leergeräumte Verzeichnis "Mond":

rmdir Mond

Der Versuch mit rmdir anders auch gleich das Verzeichnis "anders" zu löschen scheitert, da sich in "anders" noch Dateien befinden (diese möchten wir auch noch ein wenig behalten).
Links erzeugen - ln

Manchmal ist es sinnvoll, eine Datei an mehreren Orten verfügbar zu haben. Oder aber, wir wünschen die Möglichkeit, die Datei auch unter einem Namen vorliegen zu haben. Dazu erzeugen wir mit ln (link) symbolische Links:

ln -s ziel linkname

Das Verzeichnis "anders" soll nun doch auch wieder "mond" heißen:

ln -s ~/anders ~/mond

Wir probieren gleich einmal aus, dem Link zu folgen:

cd ~/mond

Hier legen wir noch rasch ein Unterverzeichnis an:

mkdir irgendwo

In "irgendwo" soll es nun einen Link zur Datei "alt.txt" geben. Der (relative) Pfad des Ziels ist immer vom Ort des angelegten Links aus zu sehen - nicht vom aktuellen Verzeichnis. Da wir den Link in "irgendwo" anlegen wollen führt der Pfad zum Ziel ins übergeornete Verzeichnis:

ln -s ../alt.txt irgendwo

(wenn der symbolische Link dort genauso heißen soll wie das Ziel, müssen wir nur das Verzeichnis angeben)

Schon gibt es auch dort ein "alt.txt".
Wir wollen "alt.txt" aber auch unter dem Namen "neu.txt" finden können:

ln -s ../alt.txt irgendwo/neu.txt

Ohne -s aufgerufen erzeugt ln sogenannte "harte Links". Ein "harter Link" ist nicht bloß ein zusätzlicher Pfad zu einer anderswo befindlichen Datei, sondern die Datei selbst. Im Prinzip ist jede Datei ein "harter Link", wovon es auch mehrere geben kann. Jede Veränderung eines Links wirkt sich auch unmittelbar auf den anderen aus (zum Beispiel beim Umbenennen). Ob von einer (normalen) Datei mehrere "harte Links" existieren, sehen wir an der Zahl, die bei der Ausgabe von ls -l nach den "Dateirechten" angeführt ist - im Allgemeinen also "1" (z.B. -rw-r--r-- 1).

In der Praxis kommen mehrere "harte Links" nicht so häufig vor und so begnügen wir uns mit ln -s, den "symbolischen Links".

---

(3) Dateirechte

Dateirechte ändern - chmod

Unter Linux hat jede Datei (auch Verzeichnisse) einen Besitzer und bestimmte Zugriffsrechte. Wir unterscheiden zwischen Lese-, Schreib- und Ausführungsrechten. Diese können bei jeder Datei für den Besitzer selbst, für Angehörige einer definierten Gruppe und für alle anderen festgelegt werden.
Mit ls -l bekommen wir diese angezeigt. Nach dem ersten Zeichen, das für den Dateityp steht (also -, d oder l), kommen 9 Zeichen, die genau jene Zugriffsrechte angeben:

(1.Zeichen=Typ)(2.-4.Zeichen=Besitzer)(5.-7.Zeichen=Gruppe)(8.-10.Zeichen=Andere)

Es gibt jeweils:

r für Lesen (read)

w für Schreiben (write)

x für Ausführen (execute) - bzw. bei Verzeichnissen, den Inhalt auflisten

- wenn das entsprechende Recht nicht vorhanden ist

Zum Beispiel:

-rw-r--r--

für eine normale Datei, die von allen gelesen, aber nur vom Besitzer geschrieben werden darf.

drwxr-xr-x

für ein Verzeichnis, das von allen gelesen und aufgelistet, aber wieder nur vom Besitzer geschrieben werden darf.

lrwxrwxrwx

für einen symbolischen Link, den jeder auch wieder löschen darf

Dies sind die häufigsten Fälle. Gelegentlich sieht man aber auch Berechtigungen wie:

-rw-rw----

hier dürfen Besitzer und Gruppe lesen und schreiben. Anderen ist es aber nicht gestattet, die Datei auch nur zu lesen.

Auch seine Berechtigung hat:

-r--------

hier darf nur der Besitzer lesen, aber auch er hat hier keine Schreibrechte, was ihn davor bewahrt, diese Datei eines Tages versehentlich zu löschen. Wenn er die Datei doch einmal löschen will, muss er zuerst die Rechte ändern.

Dazu gibt es chmod (change modus):

chmod modus dateiname

Es gibt 2 Methoden "modus" anzugeben:

1. ugoa-Methode

u steht für den Besitzer (user)

g steht für die Gruppe (group)

o steht für Andere (others)

a steht für alle (all)

Die Rechte r (lesen), w (schreiben) und x (ausführen) können jeweils hinzugefügt (+) oder entfernt (-) werden. Wir wechseln zunächst ins Verzeichnis "anders" (so wir nicht noch dort sind - "mond" ist ja auch "anders"):

cd ~/anders

Die Rechte der Datei "alt.txt" sind jetzt wahrscheinlich -rw-r--r--. Mit ls -l können wir uns das ansehen.
Wir wollen der Datei die Ausführungsrechte für alle geben:

chmod a+x alt.txt

Ein ls -l offenbart, dass sich die Rechte auf -rwxr-xr-x geändert haben.
Nun wollen wir auch dem Besitzer die Schreibrechte nehmen:

chmod u-w alt.txt

Die Rechte der Datei sind jetzt -r-xr-xr-x.
Nun wollen wir Gruppe und Anderen die Ausführungsrechte wieder nehmen und gleichzeitig dem Besitzer die Schreibrechte einräumen:

chmod go-x,u+w alt.txt

Mit ls -l sehen wir -rwxr--r--. Wir können also u, g und o kombinieren, bzw. durch Komma getrennt, auch mehrere Änderungen beauftragen.
Zuletzt wollen wir allen alle Rechte zugestehen:

chmod a+rwx alt.txt

(es macht nichts, wenn Rechte hinzugefügt werden, die schon vorhanden sind - wir tun dies, wenn wir zuvor nicht einmal nachgesehen haben, welche Rechte die Datei wirklich hatte)

So darf nun jeder alles: -rwxrwxrwx

Häufig findet man aber auch eine andere Darstellung des "modus":

2. numerische Methode

Dabei gibt es für jedes Recht eine Zahl:

1 für Ausführen

2 für Schreiben

4 für Lesen

Für die vorhandenen Rechte werden die Zahlen addiert. Für den Besitzer, die Gruppe und Andere werden nun die addierten Zahlen zu einer 3-stelligen Ziffernfolge zusammengestellt:

644 entspricht a+r,u+w
der Besitzer darf Lesen und Schreiben (4+2), Gruppe und Andere dürfen nur Lesen (4)

755 entspricht a+rx,u+w
der Besitzer darf Lesen, Schreiben und Ausführen (4+2+1), Gruppe und Andere dürfen Lesen und Ausführen (4+1), aber nicht Schreiben.

chmod 600 alt.txt

Nur noch der Besitzer darf lesen und schreiben: -rw-------
Besitzer ändern - chown und chgrp

chown besitzer dateiname

Eine Datei oder ein Verzeichnis wird auf den neuen Besitzer "besitzer" übertragen. Auch dies geht wieder für mehrere Dateien in einem Rutsch:

chown besitzer * (nicht ausprobieren!) überträgt alle Dateien im aktuellen Verzeichnis an "besitzer".

chown -R besitzer verzeichnis (nicht ausprobieren!)

ändert den Besitzer gleich auch in allen im Verzeichnis befindlichen Dateien und Unterverzeichnissen auf "besitzer".

Aber: Ich kann nicht verschenken, was mir nicht gehört. chown darf ich nur bei meinen eigenen Dateien ausführen (Dateien also, bei denen ich als Besitzer eingetragen bin). Dies gilt auch für das Ändern von Rechten über chmod.

Mit chown kann auch gleich die Gruppe geändert werden:

chown besitzer:gruppe dateiname (nicht ausprobieren!)
ändert den Besitzer auf "besitzer" und die Gruppe auf "gruppe".
Soll nur die Gruppe geändert werden gibt es chgrp:

chgrp gruppe dateiname

---

(4) Das System konfigurieren

Mit alias einen neuen Befehl erzeugen

Man kann der bash auch "Fremdsprachen" beibringen, indem man einen Alias erzeugt:

alias aliasname='programm (-optionen ...)'

Manch einer hat vielleicht schon in der Windows-Shell herumgetippt. Dort benutzt man dir (directory) um ein Verzeichnis detailiert aufzulisten. Wir können es einrichten, dass auch die Bash den Befehl dir versteht:

alias dir='ls -l'

(Hochkomma-Umklammerung ist notwendig, wenn der Befehl aus mehr als einem Wort besteht - d.h. Leerzeichen enthält)

Ab jetzt können wir dir eingeben, um das Verzeichnis aufzulisten:

dir

Ein so gesetzter Alias existiert nur bis zum Beenden der Shell, in der er gesetzt wurde.
Wenn man einen Alias ausdrücklich löschen will, gibt man folgendes ein:

unalias dir

Und weg ist er! Der Befehl dir wird ab jetzt mit der Meldung bash: dir: Datei oder Verzeichnis nicht gefunden quittiert.
Mit nano die Konfigurationsdatei .bashrc editieren

Der Texteditor nano ist auf den meisten Systemen schon von Anfang an an Bord. Wenn nicht, sollte er für diese Übung nachinstalliert werden. Er ist leicht und intuitiv zu bedienen (in der Fußzeile stehen die verfügbaren Kommandos, die mit [Strg]+[Taste] ausgeführt werden) und mein Lieblingseditor. Er wird aufgerufen mit:

nano dateiname

Über einen Eintrag in der Datei .bashrc im Home-Verzeichnis können Aliase auch dauerhaft eingericht werden. Dazu öffnen wir .bashrc mit dem Editor nano:

nano ~/.bashrc

Mit der Pfleiltaste ↓ gehen wir bis ans Ende der Datei.
Dort fügen wir eine neue Zeile mit der Alias-Zuweisung von zuvor ein:

alias dir='ls -l'

Wir speichern mit den Tasten [Strg]+[O], gefolgt von ENTER ab und schließen den Editor mit [Strg]+[X] - fertig! Beim nächsten Aufruf der Bash steht der Befehl zur Verfügung.

Um zu sehen, welche Aliase bereits vorhanden sind, geben wir folgendes ein:

alias
Variablen definieren und mit echo ausgeben

Ein Programm, mit dem sich beliebige Inhalte auf den Bildschirm drucken lassen ist echo:

echo text

Wir testen das mal (wir setzen umklammernde Hochkommas, weil Leerzeichen im Text vorkommen):

echo "Hier sitze ich nun"

Hier sitze ich nun

Gut, das ist noch nicht so beeindruckend.
Wir können aber auch den Inhalt von Variablen ausgeben. Diese müssen wir aber zuvor noch definieren:

variable=wert

Der Aufruf der Variablen erfolgt dann immer mit einem vorangestellten $:

echo $variable

Das versuchen wir:

MeinText="Ich bin wiederverwendbar!"
echo $MeinText

Ich bin wiederverwendbar!

Ich kann aber auch Zahlen in Variablen speichern und dann mit ihnen rechnen. Wir verwenden dafür let:

let argument

Argument ist hier eine Variablendefinition oder eine Rechenoperation.
Zum Beispiel:

let A=100
let B=150
let C=$A+$B
echo $C

250

Ohne let bekommen wir ein ganz anderes Ergebnis!

D=$A+$B
echo $D

100+150

Umgebungsvariablen

Es gibt auch viele spezielle Variablen. In diesen stehen bestimmte Informationen zur Verfügung, ohne, dass sie definiert werden müssen:

echo $HOME liefert den Pfad des Home-Verzeichnisses
echo $HOSTNAME den Rechnernamen
echo $LOGNAME den Benutzernamen
echo $UID die Benutzer ID

Sie werden vor allem bei Shell-Scripts benötigt.
Eine weitere, fest definierte Variable ist $RANDOM. Dennoch kann sie sehr wohl definiert werden. Sie gibt nämlich Zufallszahlen aus und bei der Definition wird ihr ein Startwert zugewiesen:

RANDOM=123
echo $RANDOM
echo $RANDOM
echo $RANDOM

usw.

Aufruf von Programmen - der Suchpfad $PATH

Programme (im weitesten Sinne) sind ausführbare Dateien bei deren Aufruf bestimmte Funktionalitäten zur Verfügung gestellt werden. In der Bash erfolgt der Aufruf durch Angabe des Pfads der Datei.

pfad/programm (-optionen parameter)

Es kann sich beim "Pfad" wieder um einen "relativen", ausgehend vom aktuellen Verzeichnis:
verzeichnis/programm oder einen "absoluten", ausgehend vom Wurzelverzeichnis:
/verzeichnis/programm
handeln.
Auch wenn ich direkt im aktuellen Verzeichnis ein Programm aufrufen möchte, muss ich den Pfad angeben - das aktuelle Verzeichnis . wird hier benötigt:

./programm

Um das Programm ls, welches sich im Verzeichnis /bin befindet, auszuführen, müsste ich also /bin/ls aufrufen. Warum genügt es aber, nur ls aufzurufen?

Die Antwort ist in der Variablen $PATH gespeichert:

echo $PATH

gibt /usr/local/bin:/usr/bin:/bin:/usr/games oder so ähnlich aus. Dies ist der aktuelle "Suchpfad" und Programme, die in Verzeichnissen liegen, die in dieser durch ":" getrennten Liste aufgeführt sind, können direkt durch Angabe ihres Dateinamens aufgerufen werden. Die Reihenfolge legt fest, wo die Bash zuerst nachsieht. Wird in /usr/local/bin ein Programm des angegebenen Namens gefunden, wird in /usr/bin nicht mehr nachgesehen und ein vorhandenes gleichnamiges Programm bleibt unberücksichtigt.

Anders als die fest belegten Umgebungsvariablen zuvor lässt sich $PATH jedoch verändern. Wir erzeugen zunächst ein neues Verzeichnis in unserem Home-Verzeichnis und integrieren es in den Suchpfad:

mkdir ~/bin
PATH=~/bin:$PATH
echo $PATH

gibt jetzt an erster Stelle das Verzeichnis "bin" in unserem Homeverzeichnis an. Wir können dort von uns selbst erstellte Scripts ablegen und sie durch einfachen Aufruf ihres Namens ausführen ... etwas später!
Das Verzeichnis muss nicht unbedingt "bin" heißen, aber es ist eine Konvention, dass ausführbare Dateien in "bin" (binaries)

liegen.

Um zu sehen, wie sich das System ohne Suchpfad verhält, können wir $PATH einmal löschen. Um sie nachher nicht wieder neu eingeben zu müssen, speichern wir sie vorher in $Backup:

Backup=$PATH
PATH=
ls

liefert uns jetzt bash: ls: Datei oder Verzeichnis nicht gefunden. Für den Aufruf von ls benötigen wir jetzt tatsächlich den vollen Pfad:

/bin/ls

Rückgängig machen wir das mit unserem Backup:

PATH=$Backup

Wie schon bei Alias besprochen, ist auch die angepasste $PATH nur in dieser Shell verfügbar und auf deren Lebensdauer beschränkt.

Wollen wir ~/bin dauerhaft im Suchpfad haben, müssen wir auch dies in .bashrc eintragen:

nano ~/.bashrc

Fügen wir am Ende einfach diese Zeile hinzu:

PATH=~/bin:$PATH

Achtung: In einigen Systemen ist ein ~/bin-Verzeichnis schon vorgesehen und wird $PATH hinzugefügt, sobald es existiert. Nach dem Anlegen des Verzeichnisses ~/bin sollten wir also ein neues Terminal öffnen und mit echo $PATH überprüfen, ob das Verzeichnis nicht schon im Suchpfad vorhanden ist.

---

(5) Umleitungen

Ausgabe in Datei umleiten - > und >>

Manchmal wollen wir die normale Ausgabe nicht am Bildschirm sehen, sondern lieber in eine Datei schreiben.

programm (-optionen parameter) > dateiname

Wir wechseln zunächst wieder mit cd ~/anders in unser Testverzeichnis.
Dort leiten wir die Ausgabe von ls -l in eine Datei um:

ls -l > info.txt

Diese Datei können wir später nach Lust und Laune lesen, ausdrucken, oder einem Freund per Mail schicken. Allerdings funktioniert dieses Verfahren nur mit Programmen, welche die Standardausgabe (die normalerweise auf den Bildschirm druckt) für ihre Ausgabe benutzen.

Den Inhalt der Datei können wir uns mit less ansehen:

less info.txt

Wir können der Datei auch noch etwas hinzufügen. > überschreibt eine bestehende Datei, >> fügt ihr etwas hinzu:

programm (-optionen parameter) >> dateiname

Das Programm uname gibt Informationen zum Kernel (Betriebssystemkern) aus. Wir fügen diese unserer Datei "info.txt" hinzu:

uname -a >> info.txt

Jetzt stehen wir mit less am Ende der Datei "info.txt" noch Angaben zu unserem Betriebssystem.

Fehlerausgabe von normaler Ausgabe trennen

Wir erinnern uns an das Problem mit den angezeigten Fehlermeldungen von find. Wenn wir dem "Umleitungszeichen" eine Ziffer voran stellen, können wir nur normale oder nur Fehlermeldungen umleiten.

Um die Fehlermeldungen (2) nicht mehr zu sehen, können wir sie in eine ganz spezielle Datei umleiten. "/dev/null" ist ein "schwarzes Loch" und alles, was wir dort hin schreiben, ist unwiederbringlich verschwunden.

find /var -name syslog 2> /dev/null

Wir können aber auch die normalen Ausgaben (1) in eine Datei schreiben und Fehlermeldungen verwerfen:

find /var -name syslog 2> /dev/null 1> syslogpfad.txt

Ein less syslogpfad.txt überzeugt uns, dass nur der Pfad von "syslog" in die Datei geschrieben wurde.

Inhalt einer Datei für die Eingabe nutzen - <

Auch der umgekehrte Weg ist möglich: die Eingaben eines Programmes aus einer Datei entnehmen:

programm < datei

Natürlich geht das nur, wenn das Programm auf Eingaben wartet. Beispiel gefällig?

Beim Überschreiben einer Datei mit cp -i gibt es eine Rückfrage die ja oder nein erwartet.
Wir legen eine Datei mit dem Inhalt ja an:

echo ja > input.txt

ja wird hier nicht auf den Bildschirm ausgegeben, wie dies bei echo eigentlich üblich ist, sondern in die Datei "input.txt" geschrieben (siehe oben). Nun wollen wir eine bestehende Datei "datei1.txt" mit "datei2.txt" überschreiben. Zunächst benutzen wir nochmals echo um die beiden Test-Dateien zu erzeugen:

echo "Erste Datei" > datei1.txt
echo "Zweite Datei" > datei2.txt

Anführungszeichen können immer verwendet werden, sind aber nicht notwendig, wenn es sich nur um ein Wort handelt. Nun kopieren wir:

cp -i datei2.txt datei1.txt < input.txt

"datei1.txt" wird ohne Rückfrage überschrieben, das heißt, die Rückfrage kommt schon, aber wir haben sie mit dem Inhalt von "input.txt" bereits beantwortet!
Rohrleitungen zu einem anderen Programm - |

Pipes (engl., Rohrleitungen) leiten den Datenstrom eines Programms an ein anderes weiter. Der Befehl lautet:

programm1 (...) | programm2 (...)

Wir verwenden wieder die Ausgabe von ls -l, leiten aber diesmal an less weiter:

ls -l | less

Das Zeichen | macht nun die Ausgabe von ls -l zur Eingabe von less.
Auslesen und filtern - cat und grep

Zuletzt noch zu zwei Programmen, die bislang noch gefehlt haben - cat und grep:

cat dateiname

cat liest Dateien bitweise aus und gibt sie auf die Standardausgabe aus. Dies können wir benutzen, um das Ende langer Textdateien anzusehen:

cat /var/log/kern.log

Bei log-Dateien interessieren uns meist vor allem die letzten Einträge und cat spult die Datei bis zum Ende ab. Manchmal ist aber auch das zuviel und ich will nur alle Einträge sehen, welche meine NVIDIA-Grafik betreffen. Da kommt mir grep zu Hilfe:

grep (-optionen) muster dateiname

Ich kann nämlich grep verwenden, um mit der Standardausgabe eines anderen Befehls als Quelle, Zeilen mit einem bestimmten Muster herauszufiltern und auszugeben.

cat /var/log/kern.log | grep usb

Dieser Befehl liefert alle Zeilen, in denen "usb" vorkommt – allerdings nur genau so geschrieben!
Will ich auch Zeilen mit großgeschriebenen Buchstaben, benötige ich die Option -i.

cat /var/log/kern.log | grep -i usb

Jetzt erhalte ich auch Zeilen, in denen "USB" nur großgeschrieben steht. Meist steht am Anfang jedes Eintrags, welches Modul die Meldung veranlasst hat. Dies ist als "Muster" üblicherweise eine gute Wahl.

Wenn ich alle Einträge sehen will, die NICHT "usb" enthalten, kann ich diese auch ausschließen:

cat /var/log/kern.log | grep -v usb

Nun erhalte ich nur den Rest.
cat kann aber auch mehrere Dateien auf einmal verarbeiten:

cat dateiname dateiname ...

Zum Beispiel:

cat datei1.txt datei2.txt > zusammen.txt

Dabei kann auch eine Wildcard für beliebige Zeichen verwendet werden:

*cat *.txt > alle_textdateien.txt*

Und cat funktioniert nicht nur für Text Dateien:

cat musik.mp3 kopie.mp3

wobei "musik.mp3" für eine beliebige mp3-Datei steht.

Allerdings dürfen Multimediadateien so normalerweise nicht miteinander verbunden werden!

---

(6) Systemadministration

Bisher haben wir uns nur im Wirkungsfeld eines einfachen Benutzers bewegt. Wenn wir an einem System arbeiten, das von einem professionellen Administrator betreut wird, könnten wir es auch dabei belassen. Auf dem eigenen PC müssen wir aber wohl selbst Hand anlegen und uns in den "Superuser" verwandeln.

Der Superuser - su und sudo

Der Administrator heißt auf UNIX-Systemen eigentlich "root" (engl. Wurzel), wird aber von Programmen oft auch als Superuser bezeichnet. Wir könnten uns beim Login gleich als "root" anmelden. Das wird aber nicht empfohlen, denn der Superuser ist sehr mächtig und ganz leicht kann ein kleiner Fehler zur Zerstörung des gesamten Systems führen. Deshalb melden wir uns immer als einfacher Benutzer an und werden nur im Bedarfsfall zum User "root". Das Programm su (switch user) erlaubt uns, die Identität zu wechseln:

su (-) (benutzername)

Ohne Benutzerangabe wird zu "root" gewechselt, was wahrscheinlich auch der häufigste Fall ist. Nach Absetzen des Befehls wird man zur Passworteingabe aufgefordert. Wir müssen das Administratorpasswort, welches wir bei der Installation vergeben haben, eingeben. Bei Ubuntu wird zunächst gar kein Passwort für "root" vergeben und das ist auch nicht vorgesehen. Ubuntu-User können diesen Absatz überspringen und bei sudo fortfahren:

su -

Das - sorgt dafür, dass wir eine Login-Shell öffnen, während wir andernfalls nur an Ort und Stelle als "root" weiter arbeiten. Wir erkennen den Unterschied daran, dass wir uns mit su - im Home-Verzeichnis des "root"-Users, also /root und nicht mehr im zuletzt aktiven Verzeichnis wiederfinden.
Wollen wir die Administratorrolle wieder ablegen, beenden wir die "root"-Shell (wie übrigens jede Shell) mit exit:

exit

Wenn wir gerade ein Programm kompiliert haben und unser aktuelles Verzeichnis das entsprechende Programmverzeichnis ist, müssen wir hier auch den Installationsschritt make install setzen und wollen nicht ins Home-Verzeichnis von "root". Hier wechseln wir sinnvollerweise ohne Minus:

su

Ehe wir aber mehr Sicherheit im Umgang mit der Shell erlangt haben, wollen wir noch nicht zum Üben an gobalen Systemdateien herumspielen und ansonst besteht überhaupt kein Unterschied zum normalen User. Wir beenden den Ausflug wieder mit exit.

Um etwas mit Administratorrechten ausführen zu können, ohne deshalb wechseln zu müssen, gibt es das Programm sudo. Auf manchen Systemen wird dies allerdings gar nicht installiert sein und ohne ein wenig Erfahrung sollte es vorerst auch nicht nachinstalliert werden. Bei Ubuntu ist dies allerdings das Standardmittel zur Administration:

sudo programm (-optionen ...)

Wir setzen vor den jeweiligen Befehl einfach sudo. Das geforderte Passwort ist diesmal das Benutzerpasswort!

sudo touch ~/rootdatei.txt

erzeugt in unserem Homeverzeichnis eine Datei, die "root" gehört und für die wir als normaler Benutzer keine Schreibberechtigung haben. ls -l überzeugt uns davon. Um sie zu entfernen benötigen wir wieder root-Rechte:

sudo rm ~/rootdatei.txt

Wer mit sudo was machen darf steuert die Datei /etc/sudoers, aber deren Konfiguration verschieben wir auf einen noch folgenden Teil.

---

(7) Hilfe!

Das wär's nun schon fast, allerdings gibt es noch einige ganz unverzichtbare Programme, die ich hier vorstellen will:

Programmdokumentationen - man, info und help

Wir haben bislang nur wenige Optionen verwendet und oft ist das auch nicht notwendig. Für gewisse Aufgaben ist es aber unerlässlich, mehr aus den Programmen herauszuholen. Dazu gibt es zu fast jedem Programm eine Dokumentation. Diese kann ich mir mit man (manual) anzeigen lassen kann:

man programmname

Die Dokumentation von ls erhalte ich mit:

man ls

Viele man-Seiten sind auf Deutsch verfügbar und wenigstens bei den grundlegenden Programmen, die wir hier überwiegend besprochen haben, von sehr guter Qualität. Bei spezielleren Programmen aus Fremdquellen kann die man-Seite aber auch unverständlich oder völlig unübersichtlich daherkommen, bzw. gar nicht vorhanden sein. Ein Blick hinein lohnt dennoch fast immer.

Achtung: Um deutschsprachige "manpages" für die hier besprochenen grundlegenden Programme zu bekommen, muss unter Umständen noch ein eigenes Paket (manpages-de) installiert werden.

Alternativ können wir auch info versuchen:

info (programmname)

Beispiel:

info ls

Ohne Programmangabe erhalten wir zunächst eine Übersicht, aus der wir durch auswählen mit dem Cursor und ENTER den gewünschten Eintrag angezeigt bekommen:

info

Etwas einfacher und nur für ganz grundlegenden Programme, die schon anfangs bei jeder Installation verfügbar sind, ist help:

help (programmname)

Es enthält allerdings weniger die bisher überwiegend besprochenen, ständig verwendeten Programme wie cd, ls oder cp, sondern beschränkt sich mehr auf Programme, deren Optionen auch der geübtere Anwender nicht im Kopf behält. Die Informationen sind auch bewusst knapp und sollen mehr erinnern als lehren:

help echo

Ohne Programmname aufgerufen, erhalten wir auch hier wieder eine Auflistung verfügbarer Programme (bzw. hier ganze Befehle):

help

Und help gibt es fast bei jeder Shell - sogar der Windows-Shell! Wenn wir also eimal vor einer fremden Shell sitzen und schon den dritten Versuch absolviert haben, der command not found hervorgebracht hat, lohnt praktisch immer ein help, das uns dann mitteilt, welche commands uns denn zur Verfügung stehen.

Die hier erlernten Befehle werden übrigens beinahe unverändert auf allen UNIX-Shells und den meisten anderen Shells in ähnlicher Form (nicht allerdings in der Windows-Shell!) funktionieren!!!

Nun, das war's für den Anfang. Wir beenden mit

exit