

Arbeiten mit der Shell / wie bekomme ich Hilfe zu Kommandos

Für die remote administration eines Unix/Linux-Servers ist es unerlässlich, die Kommandozeile (Shell) zu beherrschen, weil eine grafische Nutzeroberfläche (GUI) meist nicht zur Verfügung steht.

Die **Standard-Shell unter Linux ist die bash** (Bourne-Again-Shell), die sich aus der Bourne Shell (sh) entwickelt hat.

Die Admins loggen per ssh auf many.haecksen.org mit ihrem Benutzeraccount (nicht gleich als root) ein. Idealerweise ist ein ssh-Schlüssel dort hinterlegt, so daß die Passworteingabe entfällt. Gibt es keinen ssh-Schlüssel, wird das Passwort abgefragt. Dieses Verhalten ist im sshd (dazu später) konfiguriert.

Einmal auf der Shell "gelandet", ist es gut, zu wissen, daß es drei Arten von Kommandos gibt. Gibt man einen Befehl ein, so wird in folgender Reihenfolge nach dem Kommando gesucht:

- ist es ein **Shell-Alias** (also ein meist selbst konfigurierte Befehlsabkürzung oder -Abfolge)
- ist das Kommando ein **Shell-Builtin**
- oder ist das **Kommando ein Programm "im Pfad"**.

Je nachdem unterscheidet sich, wie man sich Hilfe holt:

- Bei eigenen Aliasen gibt es keine Hilfe, es sei denn, diese wurde explizit implementiert (was sich bei eigendefinierten Abkürzungen meist nicht lohnt)
- Bei Shell-Builtins bekommt man Hilfe mit **"help kommando"**
- Bei normalen Kommandos im Pfad gibt es in der Regel eine Manpage: **"man kommando"**

type / wie finde ich heraus, was ich denn für ein Kommando vor mir habe

Um herauszufinden, was wir vor uns haben, tippen wir:

```
type kommandoname
```

oder auch

```
type -a kommandoname
```

um alle Vorkommen eines Kommandos anzuzeigen.

Beispiel:

```
type cd  
cd is a shell builtin
```

Hier bekommen wir also Hilfe mit

```
help cd
```

Beispiel 2:

```
type -a ls
```

ergibt bei mir (Princess):

```
ls is aliased to `ls -F`  
ls is /bin/ls
```

Das erste ls ist also von mir selber definiert, keine Hilfe. Das zweite ist aber ein Kommando "im Pfad", also mit Angabe eines Verzeichnisses (/bin). Hier erhalte ich also Hilfe mit

```
man ls
```

Aufgabe: diese Seite anlesen. "durch" ist sportlich.

Der Pfad / die PATH Variable

Was aber hat es genau mit "dem Pfad" auf sich: **\$PATH** ist eine **Shell-Variable**. Anzeigen lässt man sich den Inhalt mit:

```
echo $PATH
```

darauf folgt eine Angabe wie z.B. diese:

```
/home/princess/bin:/client/bin:/client/sbin:/usr/local/bin:/bin:/usr/bin:/sbin:/usr/sbin:/usr/X11R6/bin:.
```

Wir sehen hier eine **Abfolge von Verzeichnissen, getrennt mit dem Doppelpunkt**. In dieser Reihenfolge wird in den Verzeichnissen nach dem gerade eingegebenen Kommando gesucht, wenn es eben nicht von der Shell kommt. Wichtiger Hinweis: am Ende befindet sich der Punkt, also das aktuelle Verzeichnis. Dieses ist in einer root-Shell in der Regel NICHT gesetzt, weil es das Ausführen von untergeschobenen Kommandos erleichtern würde.

Festgelegt wird diese Variable entweder im systemweiten Konfigurationsfile `/etc/bash.bashrc` oder im Konfigurationsfile im Homeverzeichnis des Nutzers.

Das Homeverzeichnis des Nutzers liegt auch in einer Shell-Variable: `$HOME`.

Aufgabe: gib dieses aus.

In einem eigenen Verzeichnis `/home/princess/bin` kann ich eigene Kommandos und Skripte ablegen.

Die Konfigurationsfiles von Kommandos

Typischerweise gibt es zu jedem Unix Kommando ein Konfigurationsfile, das idR. auf "rc" für "resource config" endet. Das systemweite File der bash ist oben erwähnt. Die eigene Konfiguration im Home-Verzeichnis beginnt mit dem Punkt und endet auf rc, also in dem Fall: `.bashrc`. Der komplette Pfad der Datei lautet also:

```
/home/username/.bashrc
```

Ebenso gibt es z.B. für das Mailprogramm mutt das systemweite `/etc/Muttrc` und das `.muttrc` im Home, oder für den Editor vi das systemweite `/etc/vim/vimrc` und das `.vimrc` im Home.

Bewegen in Verzeichnissen: cd

Da wir ja schon wissen, daß dies ein Shell-Builtin ist, hole Dir Hilfe zu diesem Kommando.

cd ohne weitere Argumente wechselt in das Home-Verzeichnis des Nutzers. Ein Verzeichnis im Baum "höher" erreicht man mit "**cd ..**". Für das "Springen" in Verzeichnisse gibt es grundsätzlich zwei Wege:

- **absolute** Adressierung mit vollem Pfad
- **relative** Adressierung vom aktuellen Verzeichnis aus

Das Wurzelverzeichnis eines jeden Unix heißt "/" (vorwärtsgerichteter Schrägstrich, gesprochen "root"). Unterverzeichnisse des Systems liegen darunter:

```
bin dev home lib lost+found mnt proc run srv tmp var  
boot etc initrd.img lib64 media opt root sbin sys usr vmlinuz
```

Aufgabe: wie kommt man mit ls zu diesem listing?

Die absolute Adressierung ist unabhängig von dem Verzeichnis, in dem ich mich aktuell befinde. Nach dem Login bin ich in meinem Home, also /home/username. Möchte ich mir eine Datei anzeigen lassen, z.B. die Systemweite CONfig-Datei vom vim, geht das also mit

```
less /etc/vim/vimrc
```

Wenn ich relativ adressiere, ist mein Ausgangspunkt relevant. Das aktuelle Verzeichnis lasse ich mir ausgeben mit "**pwd**" für "print working directory", falls ich unsicher bin, wo ich mich befinde.

Möchte ich nun mein Verzeichnis bin wechseln, reicht:

```
cd bin
```

Möchte ich in das Verzeichnis eines anderen Users gehen (soweit Rechte gesetzt sind), ginge das mit:

```
cd ../user2
```

also ein Verzeichnis hoch und dann der Name des anderen Users.

Aufgabe: wie hieße der absolute Pfad zum anderen user? Zum ersten Beispiel: wie hieße der relative Pfad zum vimrc vom Homeverzeichnis aus?

less / der Pager / Dateien ansehen

Bei den Konfigurationsfiles von Programmen handelt es sich um reine ASCII-(Text)Dateien, die man mit einem Pager ansehen kann. Das ursprüngliche Unix-Kommando dafür hieß "more". Unter Linux hat sich aber das in seinen Möglichkeiten umfangreichere "**less**" durchgesetzt.

Man kann nun

- in das Verzeichnis wechseln und mit "**less dateiname**" eine Datei anzeigen oder
- den vollen Pfad der Datei beim less angeben

Aufgabe: versuche beides für /etc/vim/vimrc .

Was aber passiert, wenn ich versehentlich eine Binärdatei ausgeben lassen will? **Aufgabe:** Teste es mit /bin/ls .

Natürlich kann ich vor dem Ansehen oder dem Versuch, eine Datei zu editieren (auch das geht nur mit Textdateien) herausfinden, was für eine Datei ich vor mir habe. Der Dateityp hängt unter Linux, anders als unter Windows NICHT an der Endung der Datei! Man findet den Dateityp heraus mit

file dateiname

Auch hier kann relativ oder absolut adressiert werden.

Aufgabe: führe das file-Kommando aus auf /etc/vim/vimrc, /bin/ls, eine Bilddatei, ein libreoffice-Dokument, ein PDF File....

mkdir: Verzeichnisse anlegen

Um sein Homeverzeichnis ein wenig zu strukturieren, braucht man **Verzeichnisse**.

Sinnvoll sind das bisher erwähnte Verzeichnis bin (Groß- und Kleinschreibung sind signifikant unter Unix!), sowie ein Verzeichnis tmp.

Diese kann man mit

mkdir name

anlegen.

Möchte man gleich mehrere Unterverzeichnisse mit anlegen, gibt es die Option -p:

```
mkdir -p Projekte/Adminen
```

Aufgabe: lege ein paar mögliche Projektverzeichnisse in Deinem \$HOME an. Hier kann man Namen recht frei gestalten, bei den Systemverzeichnissen (/etc, /usr, /bin, /sbin...) sollte man tunlichst nichts an den Namen ändern.

Namenswahl: welche Zeichen in Namen sind erlaubt?

Im Prinzip: alle außer ASCII Null. Auch Leerzeichen in Dateinamen sind möglich, da man auf der Kommandozeile aber das Leerzeichen braucht, um bei Kommandos Optionen und Argumente zu trennen, **rate ich von Leerzeichen in Dateinamen ab**, außer man möchte sich viel und oft die Finger brechen. (Leer- und andere Sonderzeichen können aber durch \ (backslash) maskiert werden, das macht auch die bash bei der automatischen Dateinamenergänzung mit <tab><tab>).

Unkritische Zeichen sind: Buchstaben a-z-A-z, Umlaute sollten vermieden werden, Punkt, Bindestrich, Unterstrich. Nur als erstes Zeichen sollte der Bindestrich (Minus) vermieden werden, weil damit Optionen zu Kommandos eingeleitet werden und man dann Mühe hat, solche Dateien wieder zu löschen.

Als hilfreich erwiesen haben sich auch Datummstempel für eine Art "Versionierung für Arme": man schreibt das Datum statt in der Form **DD.MM.JJJJ** (was zwar menschenlesbar ist, aber für eine Sortierung mit ls nicht geeignet) in der Form **JJJJMMDD**, evtl. noch mit Zeit: **JJJJMMDDSSMMSS**, also z.B. 20181206 für Nikolausi 2018.

Bearbeitet man Dateien und möchte im Fehlerfall auf eine frühere Version zurückgreifen können, kopiert man die Datei vor dem Editieren auf eine Version mit Datumsstempel, z.B.

```
cp .bashrc .bashrc.20181206
```

Hat man mehrere Versionen, listet ls diese dann chronologisch, weil es Integer sortiert.

Übrigens kann man **lange, sprechende Namen** wählen: **die bash ergänzt** Namen bei Eindeutigkeit, wenn man die **Tabulator-Taste** verwendet. <tab><tab> listet alle Namen bei Nicht-Eindeutigkeit. Ohne zuvor eingegebenes Kommando listet <tab><tab> alle zur Verfügung

stehenden Befehle, fragt aber bei einer größeren Anzahl vorher nach, ob man wirklich alles gelistet haben möchte oder eher ein paar mehr Buchstaben zur Eindeutigkeit angibt).

rm: Dateien wieder löschen

Gelegentlich möchte man auch aufräumen und Dateien wieder löschen. Dazu gibt es

```
rm dateiname
```

In dieser Form wird die Datei dann ohne Nachfragen sofort gelöscht. Es empfiehlt sich, ein Alias (s.u.) rmi zu definieren für rm -i. Insbesondere, wenn man mehrere Dateien löschen möchte, wird dann bei jeder nachgefragt und man kann mit y oder n antworten.

Das Gegenteil bewirkt -f für force.

Leere Verzeichnisse löscht man mit

```
rmdir verzeichnisname
```

Sind die Verzeichnisse nicht leer, sollen aber gelöscht werden, verwendet man

```
rm -rf
```

VORSICHT ist geboten, hier sollte sicherheitshalber immer das Verzeichnis, in dem man sich befindet, überprüft werden (mit pwd) oder man läßt sich das aktuelle Verzeichnis immer im Prompt anzeigen (s.u., Einstellungen).

Der Editor: warum man vi(m) können muß

Eigentlich ist Unix/Linux ein Betriebssystem wo sich vieles nach gusto einstellen läßt und jeder sich die Umgebung so einrichten kann, wie es ihr beliebt.

Einige Systemkommandos jedoch rufen standardmäßig den vim als Editor auf. vim ist die neuere, besser zu bedienende Version des vi. Es lohnt sich also, diesen immer verfügbaren Editor als Standard zu erlernen.

Anleitungen finden sich zuhauf, z.B. unter <https://www.selflinux.org/selflinux/html/vim.html>

vi ist also idR. ein alias auf vim (**Aufgabe:** teste dies mit den bisher gelernten Kommandos!)

An dieser Stelle aber in aller Kürze: es gibt drei Modi des vi, zwischen denen man zu wechseln verstanden haben sollte.

Der vi wird gestartet mit

```
vi dateiname
```

sodann befindet man sich im **Bewegungsmodus**. Man KANN sich im vim zwar mit den Pfeiltasten im Text bewegen, jedoch empfiehlt es sich, die Tasten

```
hjkl
```

für

```
links, unten, oben, rechts
```

zu verwenden, weil diese, kombiniert mit einer vorangestellten Ziffer, das schnelle Springen im Text ermöglichen. Dieser Editor wurde für alte, langsame Terminals entwickelt und ist daher sehr effizient.

Ist man an die Stelle gesprungen, an der man etwas editieren will, kann man in den **Editiermodus** wechseln:

```
i   füge vor dem aktuellen Zeichen ein (insert)
I   füge am Anfang der Zeile ein
a   füge nach dem aktuellen Zeichen ein (append)
A   füge am Ende der Zeile ein
o   eröffne eine neue Zeile unterhalb der aktuellen
O   eröffne eine neue Zeile oberhalb der aktuellen
```

sodann kann Text eingefügt werden.

Löschen:

```
x   löscht das aktuelle Zeichen
dd  löscht die aktuelle Zeile
```


Aufgabe: lese in der Anleitung, die wortweise gesprungen bzw. gelöscht wird. Auch: wie löscht man mehrere Zeilen auf einmal?

Der **Editiermodus** wird beendet mit

ESC

und man ist wieder im **Bewegungsmodus**.

Zum Speichern (oder Einlesen von Dateien oder suchen und ersetzen) braucht man den Kommandomodus. In diesen wechselt man vom Bewegungsmodus mit : (Doppelpunkt). Darauf folgen Kommandos

```
:w      speichere die Datei ab
:w name speichere die Datei unter einem neuen Namen ab
:r datei lese eine Datei ein
:r !kommando lese den Output eines Kommandos ein, z.b. :r !date ist gut für Kommentare
:wq     speichern und verlassen (geht auch mit ZZ)
:q!     verlassen ohne Speichern
```

Revision #2

Created 10 March 2022 15:14:15 by merline

Updated 13 June 2022 11:48:29 by caro