EDV für Physiker
Physik131 – WS 14/15
Woche 03, 20.10.–24.10.2014

Philip Bechtle, Thomas Erben

13. Oktober 2014 14:47

Inhaltsverzeichnis

3. *Unix*-Kommandos, der *Unix*-Verzeichnisbaum und das *Unix*-Rechtesystem

3.1. Lernziele und Zusammenfassung dieser Einheit

In dieser Einheit nehmen wir zunächst *Unix* Befehle genauer unter die Lupe. Sie lernen, wie *Unix* Befehle aufgebaut sind und wie man sich Informationen über sie verschafft. Als nächstes betrachten wir die Organisation von Dateien und Verzeichnissen auf einem *Unix* System. Ich zeige Ihnen, wie Sie effektiv mit Dateien arbeiten und was es mit *Sonderzeichen* auf sich hat. Wir machen mit dem *Unix* Rechtesystem weiter und schliessen diese Einheit mit Verfahren zum effektiven Arbeiten mit der *Unix* Shell ab.

3.2. Bemerkung zu *Unix Shells*

Sie haben in der letzten Einheit bereits erste *Unix* Befehle in einem *Unix Terminal* eingegeben. Genaugenommen ist das *Terminal* nur die Fensterumgebung und Sie interagieren mit dem Kommandointerpreter, der sogenannten *Shell*. Und die *Shell* ist selber ein *ganz normales* Programm und nicht eine von vorne herein fest integrierte Komponente des Systems. Dies hat, wie fast überall, die wichtige Konsequenz, dass es verschiedene Shells unter *Unix* gibt, die sich in wichtigen und wesentlichen Details voneinander unterscheiden[1]. Alles in diesem Skript und in unserer Veranstaltung bezieht sich auf die sogenannte *bash* (Bourne Again Shell), welche sich unter *Linux* weitgehend durchgesetzt hat; sie ist auch auf den CIP-Pools die Standardshell für alle Benutzer. Falls Sie auf fremden Systemen arbeiten, ist es wichtig zu wissen, welche Shell Sie verwenden. Der Befehl, um das herauszufinden, ist `echo $SHELL`. Wenn Sie die *bash* benutzen, ist die Ausgabe des Befehls etwas wie `/bin/bash`.

Aufgabe 3.1
Bitte testen Sie auf Ihrer *Linux* Umgebung, die Sie sich zuhause eingerichtet haben, welche Shell Sie benutzen:

```
user$ echo ${SHELL}    # gibt die benutzte Shell auf dem Bildschirm aus
```

Es ist für unsere Veranstaltung wichtig, dass Sie mit der *bash* Shell arbeiten, also eine Ausgabe wie `/bin/bash` bekommen. Falls Sie per default eine andere Shell benutzen[2], konsultieren Sie bitte die Dokumentation Ihres Systems oder das WWW, wie Sie die *Standardshell* auf *bash* ändern können!

[1] Denken Sie sich als Analogie zu *Microsoft Windows* die beiden WWW Browser *Internet Explorer* und *Firefox*. Beide laufen in einer *Microsoft Windows* Fensterumgebung, leisten im Primzip dasselbe, aber unterscheiden sich in wichtigen Details.

[2] vor allem wenn Sie eine Shell der *C-Shell Familie*, also *csh* oder *tcsh* verwenden sollten!

3.3. Die Anatomie von *Unix*-Befehls

In diesem Abschnitt nehmen wir die generelle Syntax eines *Unix*-Befehls genauer unter die Lupe:

```
user$ ls  -l  /usr
 ^     ^   ^    \-> "argument"
 |     |   |
 |     |    \-> "option" (switch)
 |      \
  \       \-> "Kommando"
   \
    \-> "prompt"
```

- Notwendiger Bestandteil eines jeden *Unix*-Befehls ist als erstes das eigentliche *Kommando*, hier *ls* (*list*), welches Dateien und Verzeichnisse auflistet. *Unix*-Kommandos sind üblicherweise Abkürzungen für englische Wörter oder kurze englische Sätze.
- Dem Kommando können (oder müßen) *Optionen* folgen, die das Kommando beeinflussen, hier `-l` (*long listing*). Optionen fangen mit einem − oder −− an.

  Aufgabe 3.2
  Geben Sie nacheinander die Befehle

  ```
  user$ ls /usr
  user$ ls -l /usr
  user$ ls -r /usr
  user$ ls --reverse /usr
  user$ ls -l -r /usr
  user$ ls -lr /usr
  ```

  ein. Was beobachten Sie? Wofür könnten die Optionen `-l` und `-r` stehen?

  Einbuchstabige Optionen lassen sich gut merken, wenn Sie sich bewusst machen, dass sie für ein englisches Wort stehen, das ihre Funktion beschreibt. Siehe auch Tabelle 3.1, welche gängige Optionen von Befehlen listet, denen Sie noch begegnen werden. Wie Sie in Aufgabe 3.2 gesehen haben, können Optionen kombiniert werden (`ls -l -r`), Sie können mehrere einbuchstabige Optionen unter einem - zusammenfassen (`ls -lr`) und viele einbuchstabige Optionen haben auch eine *Langform*, die durch −− gekennzeichnet ist (`ls -r` und `ls −−reverse`).
- Neben den Optionen können (oder müßen) einem *Unix*-Kommando noch Argumente folgen. Diese spezifizieren, worauf sich das Kommando bezieht. Im Beispiel `ls /usr` soll sich `ls` auf das Verzeichnis `/usr` beziehen. Sie haben bereits in der letzten Einheit gesehen, dass sich `ls` (ohne Argumente) auf das Verzeichnis bezieht, in dem Sie sich gerade befinden.
- Um Informationen und Hilfe zu *Unix*-Kommandos und ihren Argumenten zu bekommen, gibt es den Befehl *man* (*manual*) mit dem Kommando, über das man Hilfe benötigt, als Argument:

`-A`	almost all (*ls*)
`-a`	all (*ls*, *type*)
`-f`	force (*mv*, *cp*); follow (*tail*)
`-i`	interactive (*cp*, *mv*)
`-k`	key (*sort*)
`-l`	long format (*ls*)
`-n`	number of lines (*tail*, *head*); numeric (*sort*)
`-r`	reverse (*ls*, *sort*), recursive (*rm*, *cp*)
`-R`	recursive (*ls*, *chmod*)
`-t`	time (*ls*)
`-v`	verbose (*cp*, *rm*, *mv*), vice-versa (*grep*)

Tabelle 3.1.: Die Bedeutung gängiger einbuchstabiger *Unix* Optionen

```
user$ man whoami
```

Nachdem Sie das *man* Kommando ausführen wird ein sogenannter *Pager* (ein simples Textbetrachtungsprogramm) gestartet. Er stellt einfache Funktionen zum Scrollen durch den Text und zur Textsuche zur Verfügung. Nach Aufruf von *man whoami* sehen Sie etwa folgendes:

```
WHOAMI(1)                        User Commands                        WHOAMI(1)

NAME
       whoami - print effective userid

SYNOPSIS
       whoami [OPTION]...

DESCRIPTION
       Print  the  user  name  associated  with the current effective user ID.
       Same as id -un.

       --help display this help and exit

       --version
              output version information and exit
.
.
.
```

Die *Unix man-pages* sind standardisiert und enthalten alle dieselben Abschnitte (`NAME` (Programmname), `SYNOPSIS` (Aufrufsyntax), `DESCRIPTION` (Programm- und Optionenbeschreibung) und einige andere, die für Sie anfangs unwichtiger sind). Ihr Pager zum Betrachten der man-pages ist höchstwahrscheinlich *less*[3] und seine allerwichtigsten Tastaturkommandos sind:

[3]Der früher meistbenutzte Pager hatte den Namen *more*. Dieser wurde signifikant weiterentwickelt und, weil man ihm einen

SPACE	nächste Seite
b	vorherige Seite
q	Pager beenden
h	Anzeigen der Hilfe

Man-pages sind üblicherweise *komplett*, aber oft `sehr` knapp gehalten. Dies macht es Anfängern oft etwas schwer sie komplett zu verstehen. Für Sie am wichtigsten sind anfangs sicherlich die Funktionen von Programmoptionen. Diese stehen ganz am Anfang der Seiten und Sie sollten keine allzu großen Probleme haben einen Startpunkt für eine Problemlösung zu finden.

Viele *Unix*-Kommandos geben auch mit der Option `--help` Informationen:

```
user$ ls --help
```

Wie überall hilft Ihnen natürlich auch *Dr. Google* bei allen Problemen mit *Unix*-Kommandos weiter!

Aufgabe 3.3

1. Erstellen Sie mit *nano* eine Textdatei mit dem Inhalt:

   ```
   Martha      Schmitz    100
   Bernhard    Bucka      2000
   Ulf         Klein      500
   Kerstin     Meier      3800
   Achim       Zeitler    14
   ```

   und speichern Sie sie unter dem Namen `namen.txt` ab.

2. Geben Sie den Befehl

   ```
   user$ sort namen.txt
   ```

   Was tut dieser Befehl?

3. Sehen Sie unter der Hilfe von *sort* nach und sortieren Sie die Datei nach den Nachnamen (nach dem Inhalt der zweiten Spalte).

4. Sortieren Sie den Inhalt von `namen.txt` nach der dritten Spalte. Was passiert hier? Falls das Ergebnis nicht Ihren Erwartungen entspricht, konsultieren Sie erneut die Hilfe von *sort*.

Aufgabe 3.4

neuen Namen geben wollte, hat man *less* genommen. Etliche *merkwürig anmutende* Namen von *Unix* Programmen kamen auf ähnliche Weise zustande. Sie werden z. B. ein Programm namens *cat* (*concatenate*) kennenlernen, welches einfach einen Text auf dem Bildschirm ausgibt. Als man auch Text *in umgekehrter Reihenfolge* ausgeben wollte, hat man ein Programm namens *tac* geschrieben.

1. Sie wollen Hilfe über das Kommando *man* bekommen und wissen welche Optionen Sie ihm übergeben können. Wie lautet der entsprechende Befehl hierzu?
2. Sie möchten wissen, welche Befehle es in *Unix* gibt, um Dateien und Verzeichnisse zu komprimieren[4]. Sie würden hierzu gerne wissen, in welchen Dokumentationen (*man*-pages) das Wort *compress* vorkommt. Ist diese Aufgabe mit *man* lösbar (Gibt es hierzu eine Option für *man*)?

3.4. Das *Unix*-Dateisystem

Als nächstes sehen wir uns genauer an, wie *Unix* Dateien und Verzeichnisse verwaltet. In Abb. 3.1 sind typische Verzeichnisstrukturen von *Microsoft Windows* und *Unix* abgebildet. In beiden Systemen sind Verzeichnisse in einer hierarchischen, baumartigen Struktur realisiert. Ein solcher Baum hat eine *Wurzel* und darunter verzweigt sich eine astartige Struktur auf mehreren Ebenen (Verzeichnisse können wieder Verzeichnisse enthalten usw.). Der Hauptunterschied zwischen *Microsoft Windows* und *Unix* ist, dass es unter *Microsoft Windows* mehrere isolierte Bäume gibt. Jedes Laufwerk, wie die verschiedenen Festplatten, ein DVD-Laufwerk, evtl. über USB angeschlossene externe Festplatten oder Memory Sticks sind hier in einem eigenem Baum untergebracht, dessen *Wurzel* jeweils durch einen Grossbuchstaben beginnend bei `C:` gekennzeichnet ist. Unter *Unix* besteht die gesamte Verzeichnisstruktur aus einem einzigen Baum, dessen Wurzel mit `/` (Slash) gekennzeichnet ist[5].

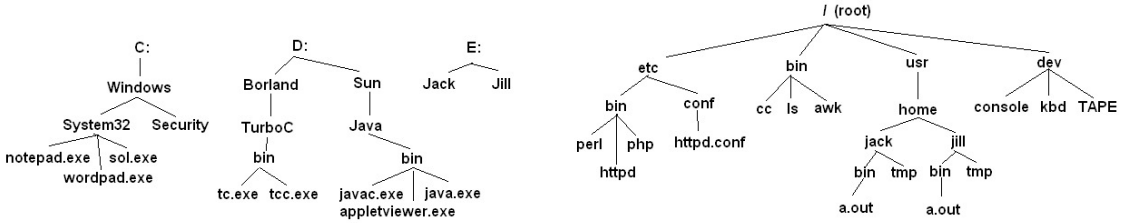

Abbildung 3.1.: Gezeigt sind typische Verzeichnisstrukturen unter *Microsoft Windows* (links) und *Unix* (rechts).

1. Man sagt, dass ein Verzeichnis *B*, das sich innerhalb eines Verzeichnisses *A* befindet, ein *Unterverzeichnis* von *A* ist. Andererseits ist *A* das *Elternverzeichnis* von *B*. Man beachte, dass *jedes* Verzeichis, ausser der Wurzel, *genau ein* Elternverzeichnis besitzt! Ich habe in der Vergangenheit gesehen, dass etliche Probleme, die *Microsoft Windows* Benutzer mit dem *Unix*-Dateisystem haben, darauf beruhen, dass sie sich der Baumstruktur nicht bewusst sind[6]. Machen Sie sich z. B. klar, dass man in Abb. 3.1 (rechts), um vom Verzeichnis `dev` zum Verzeichnis `etc` zu gelangen, einen Schritt nach oben zur Wurzel und von dort wieder einen Schritt nach unten machen muss. Man kann nicht *direkt* von `dev` nach `etc` gelangen!

[4] Sie kennen unter *Microsoft Windows* hier die sogenannten *zip* Dateien.

[5] Man kann auch argumentieren, dass *Microsoft Windows* nur einen Verzeichnisbaum hat dessen eindeutige Wurzel der *Desktop* ist!

[6] Unter *Microsoft Windows* werden diese *Details* durch klickbare Ordner und die Möglichkeit Dateien mit der Maus beliebig zwischen Ordnersymbolen hin- und herzuziehen, vor dem Benutzer verborgen.

2. Um herauszufinden, in welchem Verzeichnis man sich gerade befindet, dient der Befehl *pwd* (*present work diretory*)

   ```
   user$ pwd
   /home/thomas
   ```

   `/home/thomas` ist mein Heimatverzeichnis. Unter *Unix* trennt der Slash / verschiedene Ebenen der Verzeichnisstruktur[7]. Hier ist also `home` ein Unterverzeichnis der Wurzel und `thomas` wiederum ein Unterverzeichnis von `home`. Der gesamte Satz aus / und Verzeichnisnamen, der letztendlich zu einem bestimmten Verzeichnis führt, wird als *Pfad* bezeichnet. `/home/thomas` ist also der Pfad zu meinem Heimatverzeichnis.
   Hinweis: Eine häufige Fehlerquelle unter *Unix* ist, dass man sich nicht in dem Verzeichnis befindet, in dem man glaubt zu sein. Speziell, wenn Sie später mit vielen Terminals gleichzeitig arbeiten, verliert man schnell den Überblick, *wo* man gerade ist. Gewöhnen Sie sich am besten an, sich mit *pwd* zu vergewissern, bevor Sie lange und komplexe Programme starten, für die das Verzeichnis essentiell ist![8]

3. Um im *Unix* Verzeichnisbaum zu navigieren bedient man sich des Befehls *cd* (*change directory*):

   ```
   user$ pwd
   /home/thomas
   user$ cd /tmp
   user$ pwd
   /tmp
   ```

   Man beachte, dass *cd* keine Bestätigung über den Verzeichniswechsel gibt (ruhiges Betriebssystem; siehe Abschnitt 1.4)! Um in das Verzeichnis `/home/thomas/Vorlesung` zu wechseln kann ich neben dem einfachen

   ```
   user$ cd /home/thomas/Vorlesung
   ```

   auch in zwei Schritten gelangen:

   ```
   user$ cd /home/thomas
   user$ cd Vorlesung
   ```

   Beginnt das Argument von *cd* nicht mit dem Slash, so bezieht er sich *relativ* auf das Verzeichnis, in dem man sich gerade befindet! Es gibt zwei Verzeichnisse, in die jeder Benutzer **sehr** häufig wechseln wird und deswegen hat *cd* besondere Argumente hierfür. Dies ist einerseits das Heimatverzeichnis:

   ```
   user$ pwd
   /tmp
   user$ cd ~
   user$ pwd
   /home/thomas
   ```

   in das man mit dem ~ Argument wechseln kann[9]. Das zweite ist das Elternverzeichnis des *pwd*:

[7] Unter *Microsoft Windows* ist dies der Backslash „\“! Mein Heimatverzeichnis dort sieht wie `C:\Benutzer\Thomas` aus.
[8] Deshalb ist es auch eine sehr gute Idee, seinen Eingabeprompt so zu konfigurieren, dass er das *pwd* mit anzeigt!
[9] `cd` ohne Argument wechselt ebenfalls in das Heimatverzeichnis.

```
user$ pwd
/home/thomas/Vorlesung
user$ cd ..
user$ pwd
/home/thomas
```

in das man mit `cd ..` gelangt! Man beachte, dass **~** und **..** auch mit *allen* anderen Befehlen, die ein Verzeichnis verlangen, funktioniert. Zum Beispiel `ls ..`, `ls ~`

4. Um in *Unix* neue Verzeichnisse anzulegen dient der Befehl *mkdir* (*make directory*), und um *leere* Verzeichnisse wieder loszuwerden benutzen wir *rmdir* (*remove directory*)[10].

```
user$ pwd
/home/thomas/
user$ mkdir Test
user$ cd Test
user$ pwd
/home/thomas/Test
user$ cd ..
user$ rmdir Test
```

Ein wichtige Bemerkung ist hier die Namensgebung unter *Unix*. Obwohl es möglich ist, sollten Sie bestimmte Zeichen `nicht` in *Unix* Datei- und Verzeichnisnamen verwenden. Vermeiden Sie vor allem das unter *Microsoft Windows* populäre Leerzeichen (<space>). Sie wissen bereits, dass das Leerzeichen bei Kommandos zur Trennumg von Kommando, Optionen und Argumenten dient. Sie können sich also leicht vorstellen, dass es bei einem Verzeichnis mit dem Namen `Vorlesung␣1` gewisse Probleme gibt, wenn Sie es mit `rmdir Vorlesung␣1` loswerden wollen. Falls Sie einen Worttrenner in Datei- oder Verzeichnisnamen wollen, so verwenden Sie z. B. den *Underscore* „_“. So ist anstatt dem Verzeichnisnamen `Vorlesung␣1` die Alternative `Vorlesung_1` eine deutlich bessere Wahl. Sie wissen bereits, dass das − eine Option signalisiert und daher sind Datei- und Verzeichnisnamen, die mit − anfangen, auch eine schlechte Idee. Es gibt noch eine ganze Reihe anderer Zeichen, die unter *Unix* eine spezielle Bedeutung haben und daher in Datei- und Verzeichnisnamen nicht auftauchen sollten. Welche das sind, schreibe ich weiter unten. Ihre genaue Bedeutung werden Sie nach und nach kennenlernen. Wie in Abschnitt 1.4 erwähnt, sollten Sie unter *Unix* auch von deutschen Umlauten in Dateinamen Abstand nehmen.

5. Die letzten verbleibenden, wichtigen Operationen mit dem *Unix*-Dateisystem sind das Kopieren, Umbenennen und Entfernen von Dateien und Verzeichnissen. Hierzu stehen die Befehle *cp* (*copy*), *mv* (*move*) und *rm* (*remove*) zur Verfügung. Alle Kommandos akzeptieren Dateien und/oder Verzeichnisse als Argumente und die Bedeutung sollte jeweils intuitiv klar sein. Ich gebe unten etliche Beispiele, aus denen die Benutzung der Kommandos hervorgeht. Die meisten Schwierigkeiten mit diesen Befehlen beruhen auf Verständnisproblemen mit dem *Unix* Verzeichnisbaum (siehe oben).

```
user$ cp datei.txt datei.txt.copy  # Anlegen einer Kopie von datei.txt. Die
                                   # Kopie heisst datei.txt.copy und ist im pwd.
user$ mv datei.txt.copy test.txt   # benennt datei.txt.copy in test.txt um
user$ cp test.txt ..               # kopiert test.txt ins Elternverzeichnis
user$ cp test.txt /home/thomas/Vorlesung  # klar, oder?
```

[10] Wir sehen weiter unten, wie man Verzeichnisse mit Inhalt loswerden kann.

```
user$ mv test.txt ..                # verschiebt test.txt ins Elternverzeichnis
user$ mv ../test.txt /home/thomas/Vorlesung # Fragen?
user$ mkdir Uebung
user$ mv Uebung Uebung_1            # benennt Verz. Uebung in Uebung_1 um;
user$ mv Uebung_1 ~                 # sollte auch klar sein; aller Inhalt
                                    # in Uebung_1 wird mitverschoben
user$ rm /home/thomas/Vorlesung/test.txt # entfernt Datei test.txt aus dem Verz.
                                    # /home/thomas/Vorlesung
user$ rm -r ../Uebung_1             # entfernt REKURSIV Verzeichnis ../Uebung_1
                                    # samt allen Inhalts!!! (VORSICHT!!)
```

Man wird sehr häufig in das *pwd* kopieren wollen. Ähnlich zu der Abkürzung „`..`“, die für das Elternverzeichnis steht, gibt es eine Abkürzung für das *pwd*, nämlich den einfachen Punkt „`.`“.

```
user$ cp /home/thomas/datei.txt .  # Kopiert datei.txt aus /home/thomas ins pwd
```

Einige weitere Bemerkungen:

a) **WICHTIG:** Alle Datenmanipulationsbefehle überschreiben existierende Dateien oder löschen diese, wenn man sie dazu auffordert. Ein Nachfragen findet *in der Regel nicht statt*. Man kann ein Nachfragen bei diesen Kommandos erzwingen, indem man die Befehle mit der Option `-i` oder `--interactive` aufruft. Besondere Vorsicht ist bei dem Befehl `rm` in Verbindung mit der Option `-r` (*recursive*) geboten, wenn man ihn auf Verzeichnisse anwendet. Er löscht das Verzeichnis samt allen Inhalts, also auch eventuell vorhandene Unterverzeichnisse!

b) Der Befehl

```
user$ cp test.txt Test
```

(ähnliches gilt für `mv`) kopiert entweder die Datei `test.txt` in die Datei `Test`, *oder* er kopiert die Datei in ein *vorhandenes* Verzeichnis `Test`. Dort hat die kopierte Datei dann wieder den Namen `test.txt`! Möchte man für *Unix betonen*, dass man in ein Verzeichnis `Test` kopieren möchte, so kann man den Befehl

```
user$ cp test.txt Test/
```

geben. Das abschliessende „/“ sagt *Unix*, dass man wirklich ein Verzeichnis meint. In diesem Fall würde es eine Fehlermeldung geben, falls das Verzeichnis `Test` nicht existiert!

Das sollte erst einmal genügen, damit Sie grundsätzlich mit dem *Unix*-Dateisystem arbeiten können. Eine Kurzzusammenfassung dieses wichtigen Abschnitts mit einigen Zusatzbemerkungen finden Sie noch in Anhang B.1 dieses Skripts. Er kann Ihnen, vor allem bei Ihren ersten Schritten, als Referenz dienen.
Bevor wir mit den *Wildcards* weitermachen einige Übungen:

Aufgabe 3.5
Man bezeichnet allgemein Pfade, die mit einem „/“ beginnen als *absolute* Pfade (z.B. `/home/thomas/Vorlesung`), und solche, die *nicht* mit einem „/“ beginnen als relative Pfade (z.B. `thomas/Vorlesung`). Erklären Sie *schriftlich* was damit gemeint ist.

Aufgabe 3.6
Angenommen, Sie befinden sich mit einem neuen Account auf einer *Unix* Maschine und Sie sind sich nicht sicher, ob Sie mit einer neu geöffneten Shell in Ihrem Heimatverzeichnis sind. Wie bekommen Sie den *absoluten* Pfad Ihres Heimatverzeichnisses heraus (Angabe einer Befehlsfolge)?

Aufgabe 3.7
Im Folgenden sei `/home/thomas/` das Heimatverzeichnis des Benutzers `thomas`. Was ist die Ausgabe des jeweils letzten *pwd* Befehls in:

1. ```
   user$ pwd
   /home/thomas/Vorlesung
   user$ cd ../..
   user$ pwd
   ```
2. ```
   user$ pwd
   /home/thomas/Vorlesung
   user$ cd ../../../home/thomas
   user$ pwd
   ```
3. ```
   user$ cd ~/Vorlesung
   user$ pwd
   ```

Sind all die *cd* Befehle sinnvoll? Geben Sie gegebenenfalls kürzere Befehle mit dem selben Effekt an.

Aufgabe 3.8
Im Folgenden bezeichne `HOME` ihr Heimatverzeichnis.

1. Üben Sie die *Unix*-Kommandos *mkdir* und *cd*, indem Sie die Verzeichnisstruktur

   ```
   HOME -> hausaufgabe_1 -> ordner_1 -> ordner_3
                                     -> ordner_4
                         -> ordner_2 -> ordner_5
                                     -> ordner_6
   ```

   erstellen. Hierbei bezeichnet der `->` Unterverzeichnisse. `ordner_1` und `ordner_2` sind also Unterverzeichnisse von `hausaufgabe_1` etc. Überprüfen Sie ihre Struktur mit dem *ls* Befehl!

2. Ordnen Sie mit geeigneten *mv* Befehlen ihre Struktur in

   ```
   HOME -> hausaufgabe_1 -> ordner_1 -> ordner_2 -> ordner_3
                                                 -> ordner_4
                                                 -> ordner_5
                                                 -> ordner_6
   ```

   um. Entfernen Sie danach mit dem *rmdir* Befehl die Verzeichnisse `ordner_5` und `ordner_6`.

3. Mit dem Befehl *touch* können Sie eine leere Datei erzeugen. Erzeugen Sie eine solche namens `test_1.txt` im Verzeichnis `ordner_4`; gehen Sie also zunächst in `ordner_4` und geben Sie dort `touch test_1.txt` ein. Kopieren Sie diese Datei danach in die Ordner `ordner_3` und `ordner_2`. Überprüfen Sie mit dem *ls* Befehl, ob sich die Datei auch tatsächlich in allen drei Ordnern befindet.
   ```

Aufgabe 3.9
Jemand möchte wissen, warum es eine schlechte Idee ist, Dateien mit gewissen Sonderzeichen zu benutzen. Er gibt den Befehl:

```
user$ touch "Ein Lied.mp3"
```

Geben Sie diesen Befehl ein und überprüfen Sie mit *ls* welche Datei erzeugt wird. Löschen Sie die Datei wieder mit *rm*. Eventuell müßen Sie die *Unix* Hilfe oder das WWW zu Rate ziehen. Versuchen Sie danach mit dem *touch* Befehl eine Datei mit dem Namen `$PATH` zu erzeugen, diese in ein anderes Verzeichnis zu verschieben und anschliessend wieder zu löschen.

3.5. Wildcards

Oft möchte man viele Dateien auf einmal ansprechen, z. B. Sie möchten von all Ihren *ASCII* Textdateien in einem Verzeichnis eine Sicherheitskopie anlegen. Mit Ihrem bisherigen Wissen könnten Sie dies folgendermaßen lösen:

```
user$ ls
datei_1.txt  datei_2.txt   datei_3.txt
user$ mkdir ~/copy
user$ cp datei_1.txt datei_2.txt datei_3.txt ~/copy
```

Das Ganze wird doch recht viel Tipparbeit, wenn es noch mehr Dateien werden, die man kopieren möchte. *Unix* bietet hierfür sogenannte *Jokerzeichen* oder *Wildcards* an, um viele Dateien mit ähnlichem Namensmuster gleichzeitig anzusprechen. Das wichtigste dieser Joker ist der „*“. Er ist ein Ersatz für kein, ein oder beliebig viele beliebige Zeichen:

```
user$ ls
datei_1.txt datei_2.txt datei_3.txt
user$ mkdir ~/copy
user$ cp *.txt ~/copy    # kopiert alle Dateien, die auf '.txt' enden.
```

Jokerzeichen können *überall* verwendet werden, wo Dateinamen auftauchen. Ein paar weitere Beispiele zur Verdeutlichung und zur Anwendbarkeit:

```
user$ ls *ea*   # liste alles was irgendwo im Namen die Buchstabenkombination
                # ea besitzt.
user$ ls ea*    # liste alles was mit der Buchstabenkombination 'ea' anfaengt.
user$ ls *ea    # liste alles was mit der Buchstabenkombination 'ea' aufhoert.
user$ cp * ~    # kopiere 'alle' Dateien aus dem pwd ins Heimatverzeichnis
user$ rm *      # VORSICHT: loescht ALLE Dateien im pwd!!
```

Als nächste Wildcard steht das „?“ zur Verfügung. Es ersetzt **genau ein beliebiges** Zeichen:

```
user$ ls
datei_1.txt datei_2.txt datei_10.txt
user$ mkdir ~/copy
user$ cp datei_?.txt ~/copy    # kopiert die Dateien datei_1.txt und datei_2.txt
user$ cp datei_??.txt ~/copy   # kopiert die Datei datei_10.txt
```

Es gibt noch weitere Wildcards, die aber weniger wichtig sind und die ich hier nicht weiter aufführe. Für Interessierte fasse ich alle existierenden Wildcards referenzartig in Anhang B.2 zusammen.

Aufgabe 3.10
Das Programm *echo* gibt *einfach* sein Argument wieder auf dem Bildschirm aus:

```
user$ echo Thomas
Thomas
user$ echo /home/thomas
/home/thomas
```

Führen Sie den Befehl

```
user$ echo *
```

in einem Verzeichnis aus, in dem sich Dateien befinden. Erklären Sie *schriftlich* was hier passiert. Befragen Sie ggf. das WWW über die *Maskierung von Unix Sonderzeichen*. Können Sie einen Weg finden mit dem Kommando *echo*, ausgeführt in einem Verzeichnis, in dem sich Dateien befinden, einen Stern zu bekommen, also:

```
user$ echo ..... # ..... ist das was Sie finden muessen
*
user$
```

3.6. Das *Unix*-Rechtesystem

Wie in Abschnitt 1.4 erläutert ist es unter *Unix* besonders wichtig zu wissen, welche Rechte man selber und auch andere an den eigenen Dateien haben. Alle relevanten Informationen über bestehende Rechte werden von *ls* mit der Option *-l* (*long listing*) angezeigt:

```
user$ ls -l vorl_02.tex
-rw-r--r-- 1 thomas users ...  vorl_02.tex
              ^       ^
              |       \
              \        - Gruppe
               - Besitzer
```

Die dritte Spalte der Ausgabe zeigt den Besitzer (hier *thomas*) der Datei und die vierte Spalte die *Gruppe* (hier *users*) zu der dieser Benutzer gehört. Akzeptieren Sie für den Moment einfach, dass jeder Benutzer zu einer Gruppe gehört. Sie können daran nichts ändern. Mögliche Gruppen sind zum Beispiel Studenten, Mitarbeiter, Professoren, Administratoren, etc. Wichtig für Sie zu wissen ist, dass man in *Unix* Dateirechte für den Besitzer (das sind Sie selber), die Benutzergruppe, und *alle anderen* getrennt vergeben kann. *Alle anderen* sind hier alle Benutzer, die weder *Sie selber* sind, noch die zu Ihrer Gruppe gehören! Die erste Spalte der Ausgabe von *ls -l* gibt Auskunft welche Rechte jede dieser drei Parteien an einer Datei oder einem Verzeichnis hat:

```
user$ ls -l vorl_02.tex /bin/ls
-rwxr-xr-x 1 root   root  ...  /bin/ls
-rw-r--r-- 1 thomas users ...  vorl_02.tex
drwxr-xr-x 2 thomas users ...  beispiele/
|\^/\^/\^/
| |  |  |
| \  \  - Rechte fuer alle anderen
\  \  - Rechte fuer die Gruppe
 \  - Rechte fuer den Besitzer
  - Dateityp (- = Datei, d = Verzeichnis)
```

Besagte erste Spalte enthält insgesamt 10 Stellen. Die erste Stelle gibt an ob es sich um eine Datei (`-`), oder ein Verzeichnis (`d`) handelt. Die restlichen neun sind als Dreierblöcke zu lesen, wobei der erste für den Besitzer, der zweite für die Gruppe und der letzte für *alle anderen* steht. In jedem Dreierblock können drei verschiedene Rechte (`r = read`, `w = write` und `x = execute`) gesetzt, oder entzogen (markiert durch ein `-`) sein. Hierbei haben die drei Rechte für Dateien und Verzeichnisse jeweils etwas unterschiedliche Bedeutungen, die in Tabelle Tabelle 3.2 aufgelistet sind. Wenn Sie mit den Rechtebedeutungen und

		Datei	Verzeichnis
r	read	Datei-Inhalt lesbar (*cat*, *less*)	Verzeichnis lesbar (*ls*)
w	write	Datei-Inhalt veränderbar	Verzeichnis änderbar (Dateien anlegen/löschen/umbenennen/verschieben) *rm*, *mv*, *cp*, *ln*, *mkdir*, *rmdir*
x	execute	Datei als Programm ausführbar	Wechsel in Verzeichnis erlaubt (*cd*)

Tabelle 3.2.: *Unix*-Rechte für Dateien und Verzeichnisse

den Beziehungen zwischen Dateien und Verzeichnissen Probleme haben, stellen Sie sich ein Verzeichnis als einen Raum vor und Dateien als Gegenstände in diesem Raum. Die Rechte an dem Raum (am Verzeichnis) spezifizieren dann, ob der Rechteinhaber sich eine Inhaltsliste des Raums verschaffen darf (`r`), ob er Gegenstände aus dem Raum herausnehmen oder hinzufügen darf (`w`) oder ob er ihn *betreten* darf (`x`). Die Rechte an einem Gegenstand besagen, ob er sich diesen *genau* ansehen (die Datei lesen) darf (`r`), ob er einen Gegenstand sogar verändern kann (`w`) und ob er einen Gegenstand *benutzen* (ein Programm ausführen) darf (`x`).

Um Rechte an Dateien zu setzen und zu verändern dient das Kommando *chmod* (*change mode*). Man kann es auf viele verschiedene Arten benutzen, aber die einfachste kann leicht anhand der folgenden Beispiele verstanden werden:

```
user$ chmod u+x program   # gebe dem Besitzer (u=user) das Recht 'x'
user$ chmod g-w program   # entziehe der Gruppe (g=group) das Recht 'w'
user$ chmod o+r program   # gebe 'allen anderen' (o=others) das Recht 'r'
user$ chmod a+x program   # gebe 'allen' (user, group und others) das Recht 'x'
```

Einige Bemerkungen:

1. Sie werden später noch genauer lernen, was es mit dem (`x=execute`) Recht für Dateien auf sich hat.
2. Schauen Sie sich die Rechte von Dateien an, die Sie von außen in Ihren *Unix* Account kopieren (Dateiübertragung von Ihrem *Microsoft Windows* Rechner über das Internet, USB-Stick, CD). Es kann hier manchmal passieren, dass z. B. *alle Rechte* anfangs gesetzt sind!
3. Beachten Sie, dass das `w` Flag des Verzeichnisses darüber entscheidet, ob eine Datei, die sich darin befindet, gelöscht werden darf! Es ist daher möglich, eine Datei zu löschen, obwohl man an ihr selbst *kein* Schreibrecht (`w`) hat (Man kann einen Gegenstand aus dem Raum nehmen und wegschmeissen, aber nicht verändern)! Man kann sich daher nicht vor dem versehentlichem Löschen einer Datei schützen, indem man an ihr das Schreibrecht entzieht.
4. Andersherum können sie eine Datei in einem Verzeichnis, für das Sie keine `w`-Erlaubnis haben, keinesfalls löschen; auch wenn sie Schreibrechte an der Datei selber haben (Sie dürfen ein Blatt Papier, das sich in dem Raum befindet, beschreiben, es aber nicht aus dem Raum entfernen)!
5. Eine andere Konsequenz ist, dass Sie Dateien anderer Benutzer (insbesondere des Superusers *root*) löschen können, falls Ihnen das Verzeichnis gehört!

Hier ein Beispiel mit typischen *Standardrechten*:

```
drwxr-xr-x 2 terben users 40 Oct 11 09:51 rechte

rechte:
total 16
-rwxr-xr-x 1 terben users 9562 Oct 11 09:51 program.sh
-rw-r--r-- 1 terben users  724 Oct 11 09:51 thomas.txt
```

Ich habe ein Verzeichnis *rechte* mit den Einstellungen `drwxr-xr-x`. Ich als Besitzer darf also den Verzeichnisinhalt *listen*, den Verzeichnisinhalt *verändern* (die Dateien `program.sh` und `thomas.txt` löschen oder neue Dateien hineinkopieren) und das Verzeichnis mit *cd* betreten. Alle anderen Parteien dürfen den Verzeichnisinhalt mit *ls* auflisten und das Verzeichnis betreten, aber *nichts* am Verzeichnisinhalt verändern! An den Dateien `program.sh` und `thomas.txt` habe ich als Besitzer alle Rechte. Ich darf mir den Dateiinhalt ansehen, die Dateien verändern (z.B. editieren), und ich darf das Programm `program.sh` ausführen (laufen lassen). Alle anderen Parteien dürfen den Dateiinhalt ansehen und das Programm ausführen. Die Dateien verändern dürfen sie nicht!

Das Ganze klingt anfangs verwirrender, als es wirklich ist. Im *Normalfall* sind die Rechte neu angelegter Dateien *vernünftig* und Sie müßen sich nicht weiter darum kümmern. Sie sollten lediglich sicherstellen, dass nur Sie selber Rechte an *vertraulichen Daten und Verzeichnissen* haben!

Aufgabe 3.11
Welche Rechte brauchen Gruppenmitglieder, damit sie sich eine Datei `testdatei.txt` aus einem Verzeichnis `rechte` *kopieren* dürfen. Geben Sie die hierfür *minimal* nötigen Gruppenrechte an dem Verzeichnis und an der Datei an. Begründen Sie Ihre Antwort. Nehmen Sie an, dass die Gruppenmitglieder wissen, dass es diese Datei in besagtem Verzeichnis gibt!

Aufgabe 3.12
Im Folgenden befinden sich in `~/hausaufgaben` die Datei `analysis_1.txt` und das Programm `getmean.sh` befinden. Sie wollen:

1. dass sich alle Benutzer die Datei `analysis_1.txt` in `~/hausaufgaben` kopieren und das Skript ausführen dürfen. Sie selber sollen sämtliche Rechte an ihren Dateien und Verzeichnissen haben.
2. dass Mitglieder ihrer Gruppe die Dateien in `~/hausaufgaben` kopieren und das Skript ausführen dürfen. Alle anderen Benutzer sollen das Verzeichnis nicht betreten dürfen. Sie selber sollen sämtliche Rechte an ihren Dateien und Verzeichnissen haben.

Geben Sie für beide Fälle die nötigen Rechte an dem Verzeichnis `~/hausaufgaben` und den beiden Dateien an. Die Rechte sind jeweils für Sie selber, ihre Gruppe und alle anderen Benutzer anzugeben. Sie können annehmen, dass alle Benutzer das Recht haben das Verzeichnis `~/hausaufgaben` zu betreten.

Aufgabe 3.13
Jedes *Unix* System hat ein Verzeichnis `/tmp`, in das *alle* Benutzer kurzzeitig temporäre Dateien schreiben dürfen. In dieses Verzeichnis darf jeder Benutzer schreiben, aber `niemand` darf Dateien eines anderen Benutzers löschen. Ist ein Verzeichnis mit diesen Rechten mit dem hier vorgestellten Rechtesystem realisierbar? Geben Sie eine schriftliche Begründung. Falls Sie zu dem Schluss *nein* kommen, informieren Sie sich im WWW, wie in *Unix* ein solches Verzeichnis realisiert werden kann.

3.7. Techniken zum effektiven Umgang mit der Shell

Im Folgenden beschreibe ich noch *kurz* einige Techniken, die Ihnen bei der effektiven Nutzung der *Unix* Shell, insbesondere beim Einsparen von *viel* Tipparbeit, helfen können.

3.7.1. Tastenkürzel auf der Kommandozeile

Beim Eintippen von Kommandos in die Shell kann es manchmal zu Tippfehlern kommen. Sie brauchen dann das bereits eingetippte Kommando nicht wieder vollständig löschen oder neu eingeben, denn es stehen Ihnen für das Editieren der Kommandozeile etliche Tastenkürzel zur effektiven Arbeit zur Verfügung. Die wichtigsten sind:

↑, ↓	Blättern in alten Befehlen
→, ←	Cursor vor und zurück bewegen
C-a	Gehe zu Zeilenanfang
C-e	Gehe zu Zeilenende
C-k	Lösche Zeile ab Cursorposition
C-l	Lösche Bildschirm
C-r	Befehlsrückwärtssuche (siehe unten)
TAB	TAB-Completion (siehe unten)

3.7.2. Die TAB-Completion

Die automatische Vervollständigung von Kommandos und Argumenten mit Hilfe der TAB-Taste ist meines Erachtens das wichtigste Hilfsmittel für ein effektives Tippen auf der Kommandozeile (Hinweis: Ähnliches funktioniert auch in der Eingabeaufforderung unter *Microsoft Windows*.) Sie wirkt bei der Vervollständigung von Kommandos, Dateinamen und Benutzernamen nach ~:

```
user$ so<TAB><TAB>     # Liste aller Moeglichkeiten
soapsuds      soelim        sort          source
socklist      soffice
user$ sor<TAB>
user$ sort               # automatisch ergaenzt
user$
user$ cd ~te<TAB>
teifler  terben   tereno
user$ cd ~ter<TAB>
terben   tereno
user$ cd ~tere<TAB>
user$ cd ~tereno/         # automatisch gesetzt
```

TAB-Completion zusammengefasst:

CMD<TAB>	Name des Kommandos CMD vervollständigen wenn eindeutig
CMD<TAB><TAB>	Liste der möglichen Kommandoalternativen
CMD NNN<TAB>	Dateinamen NNN vervollständigen
~BBB<TAB>	Benutzernamen BBB vervollständigen

Desweiteren wird die TAB-Completion in vielen Programmen unterstützt.

3.7.3. Der *History* Mechanismus

Ein weiteres Hilfsmittel, das Ihnen beim Tippen sehr viel Arbeit abnimmt, sind die verschiedenen *Shell-History* Funktionen, mit denen man einmal eingegebene Kommandos wiederholen oder in leicht veränderter Form wiederverwenden kann. Von den recht zahlreichen Möglichkeiten sind für mich bei der täglichen Arbeit die folgenden relevant (erste beschriebene Möglichkeit wird von mir am meisten benutzt):

1. Das Blättern und Editieren durch bereits eingegebene Kommandos mit den Cursortasten ↑ and ↓.

2. Die *Rückwärtssuche* in alten Kommandos mit `C-r`. Geben Sie diese Kombination auf der Kommandozeile ein, so wird die Rückwärtssuche in ihrer Befehlshistory aufgerufen. Geben Sie die ersten Buchstaben eines alten Befehls ein (z.B. *ls*), dann sucht die *bash* nach alten Eingaben, die die entsprechenden Zeichen enthalten, und schreibt sie direkt in die Eingabezeile hinter Ihrer Eingabe.

   ```
   user$ <C-r>
   user$ (reverse-i-search)‘ls’:  ls -ltr *21C*sub.fits
   ```

   Es wird der erste Eintrag angezeigt, der auf Ihre Eingabe passt. Erneutes `C-r` führt zur Anzeige weiter zurückliegender, passender Einträge.

3. Sich mit dem Befehl `history` eine Liste alter Kommandos geben lassen und Relevantes mit *Maus Copy und Paste* wiederverwenden.

4. Es gibt noch die Möglichkeit, Einträge der *History*-Liste mit dem Ausrufezeichen `!` anzusprechen und auszuführen. Zum Beispiel:

   ```
   user$ history
     .
     .
     391  ls
     392  cvs diff .
     393  emacs preprocess.c &
     394  cvs diff .
     395  uname
     396  uptime
   user$ !391
   ls
   vorl_02.tex
   user$ !cvs
   cvs diff .
   .
   .
   user$ !u
   uptime
   4:59pm  up 35 days 23:38 ...
   user$ !un
   uname
   Linux
   user$
   ```

   Entweder gibt man nach dem `!` die Nummer in der History-Liste an oder einen Teilstring, der das gewünschte Kommando *hinreichend eindeutig* identifiziert. Im obigen Beispiel spricht `!u` das letzte Kommando, das mit `u` beginnt, an, also `uptime`. Erst `!un` identifiziert eindeutig `uname`.
   Der *history* Befehl zusammengefasst:

`history`	Die letzten $HISTSIZE Kommandos nummeriert anzeigen
`!NUMBER`	Kommando mit der History Nummer `NUMBER` wiederholen
`!TEXT`	*Jüngstes* Kommando mit Textanfang `TEXT` wiederholen
`!!`	Letztes Kommando wiederholen
`!-2`	Vorletztes Kommando wiederholen etc.

3.8. Eine Bemerkung zu Dateien unter *Unix*

Wir schliessen diese Einheit mit einer Bemerkung über Dateien unter *Unix* ab. Sie wissen, dass Dateien unter *Microsoft Windows* eine Endung aus drei Buchstaben besitzen, z. B. *.pdf* für Dateien des Typs *Portable Document Format* oder *.jpg* für Bilder des *JPEG* Formats. Diese eindeutige Kennzeichnug erlaubt es *Microsoft Windows* zum Beispiel Dateien mit bestimmten Anwendungen zu verknüpfen. Dieses sehr sinnvolle Konzept hat sich inzwischen auch in der *Unix* Welt weitgehend durchgesetzt, was aber nicht immer so war. Um unter *Unix* festzustellen, *was* wirklich in einer Datei steckt, gibt es das Programm *file*. Es analysiert den *Inhalt* einer Datei, um dessen Typ zu bestimmen. Für Sie kann dieser Befehl recht nützlich sein, um festzustellen, ob eine gegebene *ASCII* Textdatei vom *Unix*, oder vom *Microsoft Windows* Typ ist; siehe auch Abschnitt 2.6. Angewendet auf die in Aufgabe 2.1 vorgestellten Dateien `ascii_dos.txt` und `ascii_unix.txt` liefert dieser Befehl:

```
user$ file ascii_unix.txt
ascii_unix.txt: ASCII text
user$ file ascii_dos.txt
ascii_dos.txt: ASCII text, with CRLF line terminator  # ASCII Datei im Windows Format
```

B. Unix Tipps

B.1. Kurzzusammenfassung – Das *Unix* Dateisystem

In diesem Anhang fasse ich den Stoff zum *Unix* Dateisystem (siehe Abschnitt 3) noch einmal mit einigen Zusatzbemerkungen zusammen. Dieser Abschnitt kann Ihnen, vor allem bei Ihren ersten Schritten, als Referenz dienen.

1. Es gibt unter *Unix* Dateien und Verzeichnisse. Die Verzeichnisse bilden einen streng hierarchischen Verzeichnisbaum. Laufwerksbuchstaben wie unter *Microsoft Windows* gibt es nicht.

2. Das Trennzeichen für Verzeichnisse ist unter *Unix* „/“ und nicht „\“ (*Microsoft Windows*).

3. Wie überall unter *Unix* ist bei Datei-/Verzeichnisnamen die Gross- und Kleinschreibung essentiell. Es kann also, im Gegensatz zu *Microsoft Windows*, *verschiedene* Dateien mit den Namen `test.txt`, `TEST.txt` oder `Test.txt` in einem Verzeichnis geben.

4. Pfadangaben, die mit dem Slash `/` beginnen, bezeichnet man als absolute Pfade. Sie enthalten den gesamten Weg von der Wurzel des Verzeichnisbaums, dem Verzeichnis `/`, bis zur angegebenen Zieldatei (oder dem Zielverzeichnis). Ein Beispiel ist `/home/thomas/test.txt`. Pfadangaben, die nicht mit `/` beginnen, sind relative Pfade, die sich auf das *pwd* des angemeldeten Benutzers beziehen. Zum Beispiel spricht `text.txt` eine Datei im cwd an, `../text.txt` eine Datei im Elternverzeichnis des cwd.

5. Dateien oder auch Verzeichnisse, die mit einem „`.`“ beginnen, sind sog. versteckte Dateien bzw. Verzeichnisse. Man macht sie mit dem Befehl *ls* in Verbindung mit der Option *-a* (*all*) sichtbar. Man nutzt verstecke Dateien und Verzeichnisse oft, um Konfigurationsdateien unterzubringen. Das Konzept gibt es auch ebenso unter *Microsoft Windows* und auch dort dient ein „`.`“ am Anfang des Namens zur Identifikation. Erkundigen Sie sich bei Interesse im WWW, mit welchen Einstellungen Sie versteckte Dateien unter *Microsoft Windows* sichtbar machen.

6. Unter *Unix* sind fast alle Zeichen in Dateinamen erlaubt. Vom System her verboten sind lediglich der Slash „/“, der als Verzeichnistrenner dient, und das Nullbyte. Jedoch sollten zur Vermeidung verschiedener Probleme (mehr dazu in der nächsten Einheit) folgende Zeichen *NICHT* in Datei- oder Verzeichnisnamen enthalten sein:
   \ ; * ? @ [ ] & ~ | ( ) { } ’ < > “ <space> <tab> <return>
   Vermeiden Sie vor allem das unter *Microsoft Windows* populäre Leerzeichen (<space>), und auch „+“ und „-“ am Anfang von Dateinamen. Ebenso sollte man von deutschen Umlauten absehen.

7. Bei den meisten Benutzern sammelt sich im Heimatverzeichnis sehr schnell „Ein Haufen Müll“ an (temporäre Dateien, WWW downloads etc.) und man weiss sehr schnell nicht mehr, welche Dateien wichtig und unwichtig sind. Zur besseren Übersichtlichkeit sollten sich im Heimatverzeichnis *nur* Unterverzeichnisse befinden, in die dann Dateien einsortiert werden. Ein Vorschlag für den Aufbau einer persönliche Verzeichnisstruktur:

```
~/bin               # Eigene ausfuehrbare Programme
~/private           # Private Dateien (NICHT lesbar fuer andere)
~/public            # Zum Austausch von Dateien mit anderen
~/uebungen          # Uebungen zur EDV
~/uebungen/uebung_1
~/uebungen/uebung_2
~/tmp               # Temporaere Dateien (jederzeit loeschbar)
~/texte             # Texte/Notizen

usw.
```

8. Hier noch tabellenartig die wichtigsten Verzeichnis und Dateibezogenen Befehle zusammen mit den wichtigsten Optionen:

   a) **Auskunftsfunktionen:**

`pwd`	Aktuelles Verz. ausgeben **(present work directory)**
`cd`	In Heimatverz. wechseln **(change directory)**
`cd ~`	In Heimatverz. wechseln (äquivalent zu `cd`)
`cd ~USER`	In Heimatverz. von Benutzer `USER` wechseln
`cd ..`	In Elternverz. wechseln
`ls`	Dateien des akt. Verz. alphabetisch sortiert ausgeben
`ls DIR`	Dateiinhalt des Verz. `DIR` auflisten
`ls -r`	Sortierreihenfolge umkehren **(reverse)**
`ls -l`	ausführliche Dateiinformation **(long)**
`ls -t`	Nach Änderungsdatum sortiert zeigen **(time)**
`ls -ltr`	Drei Optionen gleichzeitig
`ls -a`	auch versteckte Dateien/Verzeichnisse **(all)**
`ls -F`	Dateityp an Namen anhängen
`ls -1`	Eine Datei pro Zeile zeigen
`ls -R`	Inhalt von Unterverzeichnissen mit auflisten **(recursive)**

   b) **Manipulationsbefehle:**

`cp FILE NAME`	File `FILE` nach `NAME` kopieren (überschreiben) **(copy)**
`cp FILE_1 ... DIR`	Dateien nach `DIR` kopieren (überschreiben)
`cp -i FILE NAME`	Rückfrage falls `NAME` existiert **(interactive)**
`cp -r DIR DEST`	gesamten Verzeichnisbaum (unter `DIR`) nach `DEST` kopieren **(recursive)**
`mv FILE NAME`	File `FILE` nach `NAME` verschieben (überschreiben) **(move)**
`mv FILE_1 ... DIR`	Dateien nach `DIR` verschieben (überschreiben)
`mv -i FILE NAME`	Rückfrage falls `NAME` existiert **(interactive)**
`mv DIR DEST`	gesamten Verzeichnisbaum (unter `DIR`) nach `DEST` verschieben
`rm FILE`	File `FILE` löschen **(remove)**
`rm -i FILE`	Vor dem Löschen nachfragen **(interactive)**
`rm -f FILE`	Löschen *ohne* Nachfragen erzwingen (setzt Option `-i` ausser Kraft) **(force)**
`rm -r DIR`	Gesamten Verzeichnisbaum (unter `DIR`) löschen **(recursive)**
`mkdir DIR`	neues Verzeichnis erstellen **(make directory)**
`mkdir -p DIR`	Verzeichnispfad erstellen (s. u.) **(path)**
`rmdir DIR`	*Leeres* Verzeichnis löschen

Zusätzliche Bemerkungen:

a) Der Befehl `mkdir -p DIR` kann einen gesamten Pfad erstellen, z. B. erstellt `mkdir -p uebung/uebung_1` das Verzeichnis `uebung` und das Unterverzeichnis `uebung_1` in einem Rutsch. Der Befehl `mkdir uebung/uebung_1` führt zu einer Fehlermeldung falls Verzeichnis `uebung` noch nicht existiert.

b) Der Befehl `rmdir DIR` löscht nur leere Verzeichnisse. Möchte man ein Verzeichnis samt Inhalt auf einmal loswerden, so verwende man `rm -r DIR`.

c) **VORSICHT** vor dem Befehl `rm -rf DIR`. Er löscht den gesamten Verzeichnisbaum unter `DIR` ohne jegliches Nachfragen! Im Gegensatz zu Windows (wo sich „gelöschte“ Dateien zunächst im Papierkorb finden und auch wieder hergestellt werden können) sind gelöschte Dateien unter *Unix* tatsächlich gelöscht!

B.2. Kurzzusammenfassung - *Unix* Wildcards

Um verschiedene Dateien (und auch Verzeichnisse) mit ähnlichen Namen effektiv ansprechen zu können, gibt es unter *Unix* die so genannten Wildcards * und ?. * ist dabei ein Ersatz für kein, ein oder beliebig viele beliebige Zeichen. Das ? ersetzt **genau ein beliebiges** Zeichen. Es gibt noch ein zu ? verwandtes Konstrukt, welches ebenfalls für **genau ein**, aber nur vom Benutzer definierte, Zeichen steht. Die Wildcard besteht aus einem eckigen Klammerpaar in das die erlaubten Zeichen eingeschlossen sind.
Die Funktionsweise ist aus folgender Tabelle und den nachfolgenden Beispielen ersichtlich:

[abc]	eines der angegebenen Zeichen ist erlaubt
[a-c], [1-5]	ein Zeichen aus dem angegebenen Bereich des Alphabets bzw. des Zahlenbereichs
[!a-c]	**keines** der angegebenen Zeichen darf vorkommen
[^a-c]	genau dasselbe wie [!a-c]
[a-cg-i1-5]	eine Kombination erlaubter Zeichenklassen

Beispiele:

- `datei[abc].dat` passt auf die Dateien `dateia.dat`, `dateib.dat` und `dateic.dat`
- `datei[1-9].dat` passt auf die Dateien `datei1.dat .. datei9.dat`
- `datei[1-9][0].dat` passt auf die Dateien `datei10.dat, datei20.dat, datei30.dat .. datei90.dat`
- `datei[a-b1-2].dat` passt auf die Dateien `dateia.dat, dateib.dat, datei1.dat, datei2.dat`

Beachten Sie, dass jede eckige Klammer genau für **ein Zeichen** steht!

Bemerkungen:

- Gibt es keine Datei, auf die ein Wildcardmuster passt, so passiert nichts und sie erhalten eine Fehlermeldung:

  ```
  user$ ls [0,9].txt
  ls: cannot access [0,9].txt: No such file or directory
  ```

- Es gibt noch die so genannte Klammererweiterung, die ähnlich zu dem Wildcardkonstrukt mit `[]` ist. Gegeben sei ein Ausdruck der Form `prefix{string1,string2,...}postfix`. An `prefix` wird dann jede Zeichenkette `string1`, `string2` etc. nacheinander angehängt und jeweils ein neuer String gebildet. Danach wird `postfix` an jeden dieser Strings angehängt, z. B.

  ```
  user$ echo a{b,c,d}e
  abe ace ade
  user$ echo a{bb,cc,dd}e
  abbe acce adde
  ```

  Im Gegensatz zu dem Wildcard ist die Klammererweiterung *nicht* an *vorhandene* Dateinamen gebunden, sondern findet auf jeden Fall statt. Meist benutze ich sie in Befehlen zur Erzeugung neuer Dateien; in den folgenden Beispielen finde ich den dritten Befehl sehr nützlich. Er kopiert eine Datei `test.txt` zu `test.txt.copy`:

```
user$ echo {con,pre}{sent,fer}{s,ed}   # geschachtelt! schon mal gesehen?
user$ ls {0,9}.txt                # Expansion findet auf alle Faelle statt!
ls: cannot access 0.txt: No such file or directory
ls: cannot access 9.txt: No such file or directory
user$ cp test.txt{,.copy}
user$ touch datei_{0,1,2,3,4,5,6,7,8,9}.txt; ls
```