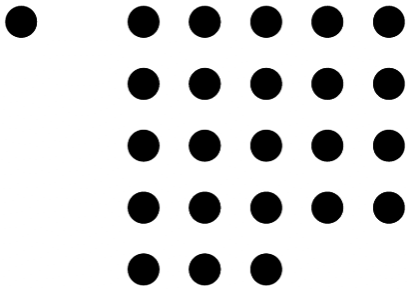
U N I X - K U R Z A N L E I T U N G



Labor für Allgemeine Datenverarbeitung
Unix Kurzanleitung für das Betriebssystem IBM-AIX V6.1
Ausgabe: Januar 2014

1 Vorwort

Diese Kurzanleitung soll als Einstieg in die Handhabung und Programmierung von IBM Power Systemen unter dem Betriebssystem AIX V6.1 dienen. AIX (**A**dvanced **I**nteractive e**X**ecutive) ist die proprietäre Version des Unix-Betriebssystems der Firma IBM.

1.1 Vorbemerkungen zu dieser Anleitung

In Unix wird grundsätzlich zwischen Groß- und Kleinschreibung unterschieden. Normalerweise werden **nur** Kleinbuchstaben verwendet.
Als Eingabeprompt erscheint auf unseren IBM-Rechnern z. B. **`advm1 /home/wi001>`**

Zur Verdeutlichung von Benutzereingaben und Systemmeldungen werden folgende Schrifttypen verwendet.

Fettdruck	*Kennzeichnet Befehlswörter, Schlüsselwörter, Dateien, Verzeichnisse und andere Elemente mit vom System vorgegebenen Namen.*
Kursivschrift	*Kennzeichnet Parameter, deren tatsächliche Namen/Werte vom Benutzer angegeben werden.*
`Monospace-Schrift`	*Kennzeichnet Beispiele für bestimmte Daten, Beispiele für am Bildschirm angezeigten Text, Beispiele für Systemnachrichten oder vom Benutzer unverändert einzugebende Daten.*

Beispiel: ***cat*** *dateiname*

Befehle (hier cat) werden immer fett und kursiv geschrieben. Variable Benutzereingaben (z.B. der Dateiname) jeweils kursiv.
Für Meldungen oder Eingabeaufforderungen des Systems wird Maschinenschrift benutzt.

```
no such file
login:
```

2 Der Einstieg

2.1 Steuertasten

Die Steuertaste ***<Ctrl>*** oder ***<Strg>*** wird zusammen mit einer anderen Taste verwendet, um Steuerzeichen einzugeben. Dabei muss die Taste ***<Ctrl>*** oder ***<Strg>*** gedrückt und gehalten und gleichzeitig eine andere Taste gedrückt werden. Einige Steuertasten werden am Bildschirm angezeigt, andere sind unsichtbar.

Die folgende Liste enthält einige nützliche Tastenfolgen und deren Funktionen.

Taste	Funktion
<Ctrl-c>	(Cancel) Beendet den aktuellen Vordergrundprozess. Am Bildschirm erscheint ^C.
<Ctrl-z>	(Suspend) Der aktuelle Vordergrundprozess wird gestoppt und in den Hintergrund der Shell gesetzt. Am Bildschirm erschein ^Z.
<Ctrl-d>	(End of File) Das Dateiendezeichen wird zum Abmelden und zum Beenden der Eingabe gesendet. Am Bildschirm erscheint ^D.
<Ctrl-u>	Die gesamte Befehlszeile wird gelöscht.
<Ctrl-s>	(XOFF) Die Bildschirmausgaben werden gestoppt.
<Ctrl-q>	(XON) Die Bildschirmausgaben werden fortgesetzt.

2.2 Passwortvergabe

Beim Einrichten einer neuen Userid wird automatisch ein Erstpasswort vergeben. Darum sollte die erste Tätigkeit nach dem Erstlogin die Vergabe eines neuen Passwortes sein. Für ein Unixsystem lautet das Kommando:

passwd

Bei diesem Vorgang muss zuerst das alte Passwort und anschließend zweimal das neue Passwort eingegeben werden, um Tippfehler zu verhindern. Es folgt der Text:

```
Old password:
New password:
Retype new password:
```

Das Passwort erscheint bei der Eingabe, wie beim Login, nicht auf dem Bildschirm.

2.2.1 Befehlseingabe

Wenn ein Benutzer einen Befehl eingibt, dann überprüft die Shell als erstes, ob sie den Befehl selbst ausführen kann. Diese Art von Befehlen bezeichnet man als **built-in-Befehle**. Wenn es kein **built-in-Befehl** ist, wird als nächstes nach einem Befehl mit einer absoluten Pfadangabe gesucht. Ist der nicht vorhanden, dann wird in den bezeichneten Verzeichnissen, die im Suchpfad (PATH) angegeben sind, gesucht. Wurde beim Aufruf des Befehls Optionen und Parameter angegeben, dann leitet die Shell diese an das entsprechende Programm weiter.

2.2.2 Aufbau von Unix Befehlen

Ein Unix-Befehl besteht aus dem Kommandonamen, Optionen und Argumenten, die, jeweils durch Leerzeichen getrennt, an den Befehl angehängt werden.

[Pfad/]Kommandoname -Option(en) Argumente

[Pfad]/]Kommandoname ist der relative oder absolute Pfad der Datei, die das Programm bzw. Script enthält. Bei built-in-Befehlen wird der [Pfad/] weggelassen.

Optionen beeinflussen die Ausführung eines Befehls. Eine Option besteht aus einem vorangestellten Minuszeichen gefolgt von einem Einzelbuchstaben. Beispiel: *ls -l* . Mehrere Optionen können oft mit einem Minuszeichen gefolgt von den Einzelbuchstaben aller gewünschten Optionen ohne weitere Zwischenräume angegeben werden. Beispiele: *ls -al* oder *ls -alg*.

Argumente sind Zeichenketten, die vom Befehl interpretiert werden.

Die Shell ist auf Leerzeichen als Trennsysmbol von Befehl, Optionen und Argumenten angewiesen. *ls-al* hat zur Folge, dass die Shell nach einem Befehl aus fünf Zeichen am Stück sucht.

Für Befehle gelten folgende Regeln:

- Leerzeichen zwischen Befehlen, Optionen und Dateinamen sind zu beachten.
- Optionen beeinflussen die Ausführungsweise eines Befehls. Optionen sind oft einzelne Buchstaben, denen ein Minuszeichen (-) vorangestellt wird. Sie werden durch Leerzeichen vom übrigen Befehl getrennt.
- Durch Verwendung eines Semikolons (;) können zwei Befehle in derselben Befehlszeile eingegeben werden. Beispiel:
  who; date
  Das Betriebssystem führt die beiden Befehle nacheinander aus.
- Bei Befehlen ist die Groß-/Kleinschreibung zu beachten. Die Shell unterscheidet zwischen Groß- und Kleinschreibung. Für die Shell sind date, DATE und Date unterschiedliche Befehle.
- Ein sehr langer Befehl kann über mehrere Zeilen eingegeben werden, indem am Ende der Zeile ein umgekehrter Schrägstrich (Backslash) eingegeben und die Return-Taste gedrückt wird. Die Fortsetzung der Eingabeaufforderung (?) wird in der nächsten Zeile angezeigt.

  Beispiel:

```
ls -al \
? > datei
```

Eine Kommandozeile wird durch Drücken der Taste Return oder Enter ausgeführt.

2.2.3 Befehle abbrechen

Wurde ein Befehl eingegeben, dessen Ausführung abgebrochen werden soll, so erreichen Sie dies durch Eingabe von ***<Strg c>*** oder ***<Ctrl c>***. Nach dem erfolgreichen Abbrechen des Befehls wird die Eingabeaufforderung der Shell (Promptzeichen) erneut angezeigt, und ein neuer Befehl kann eingeben werden.

2.3 Abmelden vom System

Nach der Benutzung des Rechners sollte man sich auf jeden Fall wieder auszuloggen, damit andere Benutzer den Account nicht missbrauchen können. Zum Abmelden genügt die Eingabe von **logout**, ***<Strg d>*** oder ***<Ctrl d>***.

3 Unix-Dateisystem

3.1 Unix-Dateiarten

Eine Datei ist aus Sicht des Benutzers im Allgemeinen eine Folge logisch zusammengehörender Informationen, aus Sicht des Systems hingegen ist sie zunächst nur eine endliche Folge von Bytes, die in Datenblöcken a 512 Byte abgelegt sind. Es geben 4 verschiedne Typen von Dateien, die für unterschiedliche Einsatzzwecke gedacht sind:

- normale Dateien,
- Dateiverzeichnisse bzw. Dateikataloge (Directories, Kataloge),
- Gerätedateien,
- Pipes.

(1) Normale Dateien
Eine normale Datei kann beliebige Daten aufnehmen, beispielsweise ablauffähige Programme, jede Form von binären Daten oder Texte im ASCII-Code. Unix erwartet keinerlei spezielle Strukturierung einer Datei in Sätze, Blöcke, Sektoren usw. Eine Textdatei besteht beispielsweise aus einer Folge von Zeichen mit einer übergeordneten Zeilenstruktur, wobei die Zeilenenden durch besondere Zeilenendezeichen vermerkt sind.

(2) Dateiverzeichnisse
Dateiverzeichnisse sind Dateien, die Verweise auf weitere Dateien enthalten. Die Verweisstruktur ist hierbei hierarchisch. Der Eintrag einer Datei in das Dateiverzeichnis besteht aus der systeminternen Knotennummer (i-node-number) und dem Dateinamen. Falls noch keine Dateien vorhanden sind, dann besteht nur ein Eintrag auf sich selbst und dem zugehörigen Vaterkatalog.

(3) Gerätedateien
Gerätedateien, auch special files genannt, sind Dateien, die die Schnittstelle zu den Peripheriegeräten darstellen, die vom Betriebssystem verwaltet werden. Jeder Lese- oder Schreibversuch auf eine Gerätedatei wird direkt auf das zugehörige Gerät weitergeleitet. Durch ihre, den normalen Dateien identische Behandlung, ergibt sich für den Benutzer kein Unterschied zwischen der Ein- /Ausgabe auf Dateien oder auf physikalischen Geräten und damit auch weitestgehende Geräteunabhängigkeit.

(4) Pipes
Die Pipe verbindet die Standardausgabe eines Prozesses mit der Standardeingabe eines anderen Prozesses.

3.2 Unix-Dateistruktur

Das Unix-Dateisystem ist hierarchisch strukturiert und hat, grafisch dargestellt, die Form eines umgedrehten Baumes. Der Ausgangspunkt des UNIX-Dateibaumes ist das Wurzelverzeichnis (root directory), das unter Unix mit /, gesprochen slash, dargestellt wird. Da jedes Verzeichnis seinerseits weitere Verzeichnisse enthalten kann, ergibt sich ein baumartiger Aufbau mit Ästen. Die Blätter entsprechen dabei Dateien oder leeren Verzeichnissen. Mit dem Kommando *mount* werden die verschiedenen Platten und Laufwerke des Systems in leere Verzeichnisse des Dateibaums eingehängt. Bei dem Wechsel in ein gemountetes Verzeichnis wechselt ein Besucher so automatisch auch das Medium, auf dem er arbeitet. Mit dem Kommando *umount* werden Platten wieder beim System abgemeldet. Will man nun eine Datei oder ein Verzeichnis in einem Programmaufruf namentlich angeben, so schreibt man dafür einfach den Datei- bzw. Verzeichnisnamen an der entsprechenden Stelle in das Kommando. Ist das nicht der Fall, so kann man dem System angeben, welcher Weg durch die Verzeichnisse zu der gewünschten Datei führt. Diese Angabe nennt man Zugriffspfad. Die Angabe des Zugriffpfades bezieht sich entweder auf das Verzeichnis, in dem man sich gerade befindet (das Arbeitsverzeichnis bzw. Working Directory oder Home Directory) oder auf das root-Verzeichnis. Entscheidend dafür ist das erste Zeichen der Pfadangabe:

- Ein / am Anfang des Pfades bedeutet, dass ab dem root-Verzeichnis gesucht werden soll.
- Jedes andere Zeichen gibt an, dass sich der Suchpfad auf das aktuelle Verzeichnis bezieht.

In dieser Baumstruktur sind spezielle Verzeichnisse vorhanden, die in erster Linie systemeigene Dateien enthalten aber auch das Verzeichnis, in dem die eigenen Daten abgelegt sind.

Diese Baumstruktur ist aber nicht bei allen Systemen identisch. Das hängt davon ab, ob es sich um ein BSD (Berkley Software Distribution der Universität von Kalifornien in Berkley) oder System V (der Firma AT&T) Derivat handelt.

Bei den IBM Power Systemen bekommt jeder Benutzer ein eigenes Loginverzeichnis für seine Daten. Das Verzeichnis lautet: /home/Benutzername. Dies bezeichnet man als das Arbeitsverzeichnis, das Working Directory oder Home Directory. Dort können dann eigene Dateien abgelegt, bzw. weitere Unterverzeichnisse angelegt werden.

3.3 Umgang mit Verzeichnissen (Directories)

Alle Dateien und Verzeichnisse müssen mit einem einheitlichen Namenszug auffindbar und zugreifbar sein. Dafür wurden folgende Vereinbarungen definiert:

- / (Slash)
  Name des Rootverzeichnisses.

- /dir/dir/.../
  als Trennzeichen zwischen Verzeichnissen und Dateien.

- .(Dot)
  symbolischer Name für das aktuelle Verzeichnis, in dem man sich gerade befindet.

- ..
  symbolischer Name des Vorgängerverzeichnisses (das Vaterverzeichnis des aktuellen Verzeichnisses).

- ~
  (Tilde) symbolischer Name für das Arbeitsverzeichnis (Home Directory) des Benutzers.

Für den Umgang mit Dateiverzeichnissen (Directories) geben es folgende Kommandos.

3.3.1 mkdir (Erzeuge ein Verzeichnis)

Will man ein neues Dateiverzeichnis (Directory) erzeugen, so benutzt man das Kommando

mkdir *Dirname*

(**m**a**k**e **dir**ectory). Es wird das neue Verzeichnis *Dirname* angelegt. Wird das Verzeichnis erzeugt, so werden automatisch die zwei Standardeinträge "." und ".." angelegt. Der Name "." ist gleichbedeutend mit dem Verzeichnisnamen selbst, und der Name ".." bezieht sich auf das unmittelbare Vorgängerverzeichnis.

Ist das aktuelle Verzeichnis */home/name*, dann erzeugt das Kommando *mkdir a1* das Verzeichnis */home/name/a1*. Mit ***rmdir*** kann man das Verzeichnis wieder löschen (siehe 3.3.2).

3.3.2 rmdir (Lösche ein leeres Verzeichnis)

Mit dem Kommando

rmdir *Dirname*

(**r**e**m**ove **dir**ectory) kann man ein leeres Verzeichnis, d.h., es dürfen nur die Einträge "." und ".." bestehen, löschen. Für das Beispiel in 3.3.1 bedeutet das, dass man sich im Verzeichnis */home/name* befinden muss, um *a1* löschen zu können.

3.3.3 pwd (Zeige das aktuelle Verzeichnis an)

Mit dem Kommando

pwd

(**p**rint **w**orking **d**irectory) kann man sich das aktuelle Verzeichnis anzeigen lassen.

3.3.4 cd (Wechsle in ein anderes Verzeichnis)

Mit dem Kommando

cd

(**c**hange **d**irectory) kann man sich in dem gesamten Dateibaum bewegen, soweit man dafür die entsprechenden Rechte besitzt. Nachfolgend eine Liste der möglichen Befehle:

cd /dirname	Gehe in das Verzeichnis *dirname*, das sich auf den Eintrag im Root-Verzeichnis bezieht.
cd dirname	Gehe in das Verzeichnis *dirname*, das sich auf den Eintrag im aktuellen Verzeichnis bezieht.
cd ..	Gehe im Dateibaum ein Verzeichnis nach oben. (Vorgänger-verzeichnis)
cd .	Gehe in das Verzeichnis, in dem man schon ist. (bewirkt also nichts)
cd	Gehe ins Homedirectory. (Working Directory)
cd ~	Gehe ins Homedirectory. (Working Directory)

3.3.5 ls (Dateiverzeichnis anzeigen lassen)

Mit dem Kommando

ls *-Optionen Name*

(**l**i**s**t) kann man sich die Dateien in einem Dateiverzeichnis (Directory) anzeigen lassen. Nachfolgend einige Optionen:

-a	Alle Dateien werden aufgeführt, auch die mit einem "." beginnen.
-d	Ist eine Datei ein Verzeichnis, wird der Name, aber nicht sein Inhalt ausgegeben.
-F	Verzeichnisse werden mit einem "/" markiert.
-l	Ausführliche Liste des aktuellen Verzeichnisses.
-i	Gibt zusätzlich die Nummern der belegten I-Nodes aus.
-t	Sortiert nach der Zeit (letzte Veränderung) anstelle des Namens.

3.3.6 Verlagern oder Kopieren von Dateien in ein Verzeichnis

Mit dem Kommando ***mv*** (**m**o**v**e) oder ***cp*** (**c**o**p**y) kann man eine Datei, mehrere Dateien oder Verzeichnisse in ein anderes Verzeichnis verlagern oder kopieren. Dies kann man erreichen durch

mv Datei1 Datei2 ... Verzeichnis

oder

cp Datei1 Datei2 ... Verzeichnis.

Die Dateien behalten in dem neuen Verzeichnis ihren ursprünglichen Namen. Das mv-Kommando verlagert oder benennt um, das cp-Kommando kopiert die Dateien in das neue Verzeichnis.

Einige nützliche Optionen sind:

-R	Rekursives Kopieren von Dateien und Verzeichnissen in das Zielverzeichnis.
-i	Falls die Zieldatei bereits vorhanden ist, wird gefragt, ob sie überschrieben werden soll.
-p	Datum und Zugriffsrechte bleiben erhalten.

Mit dem nachfolgendem Befehl kopiert man alle im Verzeichnis *Quelle* stehenden Unterverzeichnisse und Dateien in das Verzeichnis *Ziel*.

cp -R Quelle Ziel.

Bei Unix muss man sorgfältig darauf achten, ob das Ziel eine Datei oder ein Verzeichnis ist, da das Kommando ***mv*** auch Dateien löschen kann.
Beispiel:

```
advm1> mv datei1 ziel
advm1> mv datei2 ziel
advm1> mv datei3 ziel
```

Falls *ziel* **ein** Verzeichnis ist, dann bestehen nachher folgende Dateien: *ziel/datei1*, *ziel/datei2* und *ziel/ datei3*.

Falls *ziel* **kein** Verzeichnis ist, wird *datei1* in *ziel* umbenannt, dann *datei2* in *ziel* und schließlich *datei3* in *ziel*, wobei die vorherige Datei *ziel* jeweils überschrieben wird. Als letztes steht der Inhalt von *datei3* in *ziel*.

3.4 Umgang mit Dateien

3.4.1 Aufbau eines Dateinamens

Ein Dateiname darf bis zu 255 Zeichen lang sein, wobei alle Zeichen **außer** / erlaubt sind, also auch solche, die nicht druckbar sind. Allerdings können sich bei der Verwendung bestimmter Zeichen unter Umständen Probleme im Zusammenhang mit der Shell oder dem Kommandointerpreter ergeben. Man sollte sich deshalb auf solche Zeichen beschränken, die nicht zu Konflikten mit Kommandooptionen führen. Dies sind insbesondere die Zeichen *, |, <, >, &, ;, (,), [und]. Ein Leerzeichen in einem Dateinamen kann durch die Eingabe \<blank> eingegeben werden. Wenn man sicher gehen will, dann sollte man nur folgende Zeichen benutzen:

- Klein- und Großbuchstaben (Unix unterscheidet zwischen Klein- und Großbuchstaben)
- Ziffern
- Unterstrich (_)
- Punkt (.) (Dateinamen, die mit einem Punkt beginnen sind "unsichtbar", d.h., *ls* zeigt diese Dateien nicht an. Dies ist nur mit der Option *-a* möglich. In dem Dateinamen kann man beliebig viele Punkte verwenden).
- Minus (-) (Darf nicht am Anfang eines Dateinamens stehen, da man sonst mit Kommandooptionen in Konflikt geraten kann).

Die Dateinamen dürfen frei gewählt werden, aber einige Namenskonventionen finden für bestimmte Produkte Verwendung. Nachfolgend einige übliche Endungen:

.a *für Objektbibliotheken.*

.c *für C-Quelldateien.*

.cc oder .C *für C++-Quelldateien.*

.o *für Objektdateien.*

.tar *für Archiv-Dateien.*

.zip *mittels MS-DOS ZIP komprimierte Datei.*

.html *für eine Datei, die HTML-Befehle für das World Wide Web enthält.*

3.4.2 Automatische Ergänzung von Dateinamen

Die Shell kann Dateinamen automatisch ergänzen. Dafür gibt es folgende Mechanismen:

*	Eine beliebig lange Folge von Zeichen.
?	Ein einzelnes beliebiges Zeichen.
[…]	Eines der Zeichen, die in der Klammer vorkommen.
[!...]	Ein beliebiges Zeichen, ausgenommen die in der Klammer

Einige Beispiele:

*	Selektiert alle Dateien.
???	Selektiert alle Dateien oder Directories, deren Name drei Zeichen lang ist.
?w?	Selektiert alle Dateien oder Directories, deren Name drei Zeichen lang ist und an der zweiten Stelle ein 'w' enthalten.
[abd]	Selektiert alle Dateien oder Directories, die mit 'a', 'b' oder 'd' beginnen.
*.[ef]	Selektiert alle Dateien deren Extension mit 'e' oder 'f' beginnen.
[a-z]*	Selektiert alle Dateien oder Directories deren erstes Zeichen zwischen 'a' und 'z' liegt.
test[0-1][0-9]	Selektiert alle Dateien von test01 bis test19.

3.4.3 Dateizugriffsrechte und Dateiattribute

Jede Datei im Unixdateibaum besitzt Dateirechte. Bei jedem Zugriff auf eine Datei wird überprüft, ob die Zugriffsrechte dies erlauben. Doch die Zugriffsrechte alleine reichen nicht aus, um eine Datei vor unberechtigtem Zugriff zu schützen. Deshalb wird jedem Benutzer bei der Anmeldung eine eindeutige Benutzernummer (**U**ser **ID**entification UID) zugewiesen. Die Zuordnung von Namen zu Benutzernamen (Userid) wird durch die Datei */etc/passwd* vollzogen. Zusätzlich wird jedem Benutzer in dieser Datei eine Gruppennummer (**G**roup **ID**entification GID) zugewiesen. Jeder Benutzer am System ist also mit einer Benutzernummer und einer Gruppennummer ausgestattet. Unix vergisst nämlich nach der Anmeldung den Namen und arbeitet immer mit diesen Nummern. Wenn der Name benötigt wird, wird dieser jedes Mal aus der Datei */etc/passwd* extrahiert. Welche Nummern man besitzt, lässt sich durch das Kommando ***id*** ermitteln. Die Ausgabe für den Benutzer *il001* sieht folgendermaßen aus:

```
advm1> id
uid=499(il001) gid=15(labor)
```

Wenn ein Anwender nun eine Datei anlegt, so gehört diese ihm. Das heißt, er darf die Zugriffsrechte festlegen. Er kann den lesenden, schreibenden und ausführenden Zugriff auf die Datei freigeben. Diese Rechte werden durch ein r, w und x dargestellt. Ein nicht eingeräumtes Recht wird durch ein - symbolisiert.

Man kann also nicht Eigentümer einer fremden Datei werden und man kann Dateien nicht verschenken. Man kann jedoch Eigentümer einer Kopie einer fremden Datei werden.

Die Attribute (systemseitig gespeicherte Eigenschaften) einer Datei kann man sich durch das Kommando ***ls*** ansehen. Das Kommando *ls –l test1* in */home/il001* erzeugt folgende Ausgabe:

```
advm1> ls –l test1
-rwxr-xr-x   1  il001 labor       42328 Sep 10 2003 test1
```

Das erste Zeichen (in diesem Fall ein Minus) gibt das Dateiart-Flag an und die 9 folgenden Zeichen geben die Zugriffsberechtigung an. Diese 9 Zeichen unterteilen sich in 3 Benutzerklassen.
Beim Dateiart-Flag (1. Zeichen) wird folgendes angezeigt:

Datei-typ	Datei	Verzeichnis
-	file	Normale Datei.
b	block special device	Blockorientierte Geräte.
c	character spezial device	Zeichenorientierte Geräte.
d	directory	Verzeichnis.
l	symbolic link	Querverweis auf eine andere Datei.
p	pipeline[1]	FIFO-Puffer oder Named Pipe. Eine Datei, die der Kommunikation von Prozessen dient.
S	socket[2]	Eine Datei, die der Kommunikation von Prozessen über eine Netzwerkverbindung dient.

Die nächsten neun Zeichen zeigen die Zugriffsberechtigung an und werden in drei Benutzerklassen zu je drei Zeichen aufgeteilt. Die Benutzerklassen sind:

u	Der Eigentümer der Datei (user)	Zeichen 2 - 4
g	Die Mitglieder der Gruppe (group)	Zeichen 5 - 7
o	Alle anderen Benutzer des Systems (other)	Zeichen 8 - 10

Die ersten drei Zeichen zeigen die Berechtigung des Eigentümers, die nächsten drei die der Gruppe und die letzten drei die Berechtigung aller anderen Benutzer an, die auf die Datei zugreifen können. Die drei Zeichen innerhalb der einzelnen Gruppen zeigen die Lese-, Schreib- und Ausführungsberechtigung der Datei an.

Die Rechte werden folgendermaßen überprüft:

Stimmen UID des Anwenders und der Datei überein, dann gilt die erste Rechtegruppe (user).
Wenn nicht, vergleicht Unix die GIDs aller Gruppen, in denen der Anwender Mitglied ist mit der GID der Datei. Stimmt eine der GID mit der GID der Datei überein, gelten die Rechte der zweiten Gruppe (group).
Ansonsten gelten die Rechte der dritten Rechtegruppe (other).

[1] FIFO = First in, first out.
[2] Datei als Hilfsmittel zur Kommunikation zwischen Unix-Prozessen.

Die Zugriffsrechte für Dateien und Verzeichnisse haben unterschiedliche Wirkungen. Nachfolgend eine Tabelle der Zugriffsrechte:

Rechteart	Datei	Verzeichnis
r	Die Datei kann gelesen werden.	Man kann den Inhalt des Verzeichnisses lesen.
w	Die Datei darf modifiziert werden. Ob die Datei gelöscht werden kann, ist keine Eigenschaft der Datei, sondern eine Eigenschaft des Verzeichnisses, in dem sich die Datei befindet.	Man kann Dateien in dem Verzeichnis erzeugen und löschen.
x	Die Datei darf durch Aufruf ihres Namens als Programm ausgeführt werden. Dies können Binär-Programme oder Shell-Skripte sein.	Es ist gestattet mit dem Kommando cd dieses Verzeichnis zu "betreten" oder es darf die Suche nach einer Datei erfolgen.
-	Das Recht wird nicht gewährt.	Das Recht wird nicht gewährt.
s (SUID)	Das Programm läuft mit der UID des Eigentümers.	Die Angabe wird ignoriert.
s (SGID)	Das Programm läuft mit der GID des Eigentümers.	Alle Dateien, die in diesem Verzeichnis erzeugt werden, bekommen die Gruppe des Verzeichnisses.
t (SVTX)	Keine Bedeutung.	Zum Löschen der Datei in einem Verzeichnis muss der User der Eigentümer der Datei oder des Verzeichnissen sein. (Wird bei /tmp benutzt.)

Für unser Beispiel bedeutet das, dass der Eigentümer die Datei lesen, beschreiben und ausführen kann (rwx), die Gruppe kann lesen und ausführen (r-x), aber nicht schreiben. Alle anderen Benutzer können ebenfalls lesen und ausführen (r-x), aber nicht darauf schreiben. Wie man die Zugriffsrechte verändern kann, siehe Kapitel 3.4.5.

Die 1 in Spalte 2 unseres Beispiels bezeichnet die Anzahl der Links (Verweise) auf diese Datei. Dies ist eine Besonderheit des Unix-Dateisystems. Eine Unix-Datei darf mehrere Namen haben, also unter mehreren Namen in verschiedenen oder auch gleichen Verzeichnissen existieren. Der Name *il001* in Spalte 3 gibt den Eigentümer der Datei an. Der Name *labor* in Spalte 4 gibt den Gruppennamen zu der die Datei *test1* gehört an. (Falls der Eigentümer oder Gruppen-Name nicht mehr bekannt ist, dann steht an dieser Stelle die UID (user-id) bzw. die GID (group-id). Dies ist z.B. möglich, wenn zwei Rechner nur auf einem Rechner ein gemeinsames Dateisystem haben. (NFS-Cluster = Network-File-System). In Spalte 5 steht die Dateilänge in Bytes. In den nächsten 3 Spalten stehen das Erstellungs- bzw. das letzte Dateiänderungsdatum. In der letzten Spalte steht der Dateiname.

3.4.4 Erweiterte Zugriffsrechte

Bisher sind nur Lese-, Schreib- und Ausführungsrechte betrachtet worden. Das ist aber nicht die einzig mögliche Sichtweise. Es kann das berechtigte Interesse des Eigentümers einer Programmdatei sein, das der Prozess, der durch die Ausführung dieser Datei entsteht, auch sein Eigentum ist. Dies tritt dann auf, wenn ein Programm Systemdienste für alle System-Benutzer anbieten soll.

3.4.4.1 Das SUID- und SGID-Bit bei ausführbaren Dateien

Der Eigentümer einer Programmdatei kann deshalb den Ausführungsmodus einer Programmdatei so verändern, dass das Betriebssystem den daraus entstehenden Prozess mit seiner effektiven User-ID oder mit der effektiven Group-ID seiner Benutzergruppe ausführt. Der Prozess einer Programmdatei mit S-Modus hat damit die Rechte des Eigentümers (bzw. der Gruppe) der Datei. Zusätzlich behält jeder Prozess mit einer veränderten User- oder Gruppen-ID auf einer zweiten Ebene auch die realen Identitäten und Rechte der aufrufenden Person. Damit kann solch ein Prozess in beiden Bereichen arbeiten. Ein *Set User ID Programm* bildet auf diese Weise ein „intelligentes Tor" zwischen den sonst hermetisch voneinander abgeriegelten Bereichen zweier User. In der Regel wird dies bei Programmen angewandt, bei dem man Zugriff auf Ressourcen benötigt, auf die normalerweise nur *root* Zugriff hat. Z. B. das Kommando ***passwd*** hat das SUID-Bit gesetzt. Das bedeutet, dass der Befehl mit Systemverwalterrechten abläuft, weil der Eigentümer des Kommandos *root* ist.

Die Benutzung eines SUID-Programms setzt das Vertrauen des Eigentümers in die Zuverlässigkeit des Programms voraus. Besonders Programme mit der User-ID von *root* können bei Fehlfunktionen beträchtlichen Schaden an Systemdaten oder Userdaten verursachen.

Alle Programme, die im SUID-Modus gesetzt werden, müssen gut getestet, zuverlässig und sicher sein.

Sie erkennen Programme, die mit dem SUID- oder SGID-Recht gesetzt sind, bei dem Aufruf von ***ls –l***, durch die symbolische Abkürzung von s anstelle von x. (Ein großes S bedeutet, dass das SUID- oder SGID-Bit gesetzt, aber das x-Bit **nicht** gesetzt ist.)

3.4.4.2 Das SGID-Bit bei Verzeichnissen

Bei Verzeichnissen hat das SGID-Bit eine andere sehr nützliche Eigenschaft. Wenn das SGID-Bit gesetzt ist, gehören alle Dateien, die in diesem Verzeichnis erzeugt werden, derselben Gruppe an, der auch das Verzeichnis gehört. Das gilt auch für das Erzeugen neuer Verzeichnisse. Dabei wird das SGID-Bit mit vererbt.

Es ist zwar möglich, auch bei einem Verzeichnis das SUID-Bit zu setzen, diese Einstellung hat aber keine Wirkung. Das Betriebssystem erlaubt es den Usern nicht, Dateien an andere User „zu verschenken".

3.4.4.3 Das SVTX-Bit bei Verzeichnissen

Wenn mehrere Benutzer in einem Verzeichnis Schreibrecht haben, können diese Benutzer auch alle dort vorhandenen Dateien löschen. Das ist nicht immer erwünscht. Man kann dieses Problem vermeiden, wenn man den Eignern der Dateien in diesem Verzeichnis das SVTX-Recht (**S**a**v**e **T**e**x**t Bit) gibt, das auch als Sticky-Bit bzw. als T-Recht bezeichnet wird. Dieses Recht wird der Klasse der „an-

deren“ (o) zugeordnet. Dateien können jetzt nur noch vom Eigentümer oder dem Systemverwalter gelöscht werden. Dies wird besonders bei öffentlichen Verzeichnissen verwandt. Deshalb sind in öffentlich benutzbaren Verzeichnissen wie */tmp* und */var/tmp* standardmäßig die Rechte **`rwxrwxrwt`** gesetzt. Ob das Recht gesetzt ist sehen Sie durch die symbolische Abkürzung von t anstelle von x. (Ein großes T bedeutet, dass das SVTX-Bit gesetzt, aber das x-Bit **nicht** gesetzt ist.)

3.4.5 Verändern der Zugriffsrechte

Mit dem Kommando

chmod -Optionen Rechte Dateiname oder
chmod –Zahlencode Dateiname

(**ch**ange **mod**e) kann der Eigentümer die Zugriffsrechte für eine Datei bzw. ein Verzeichnis ändern. Man gibt dabei an, für wen die Rechte geändert werden sollen, nämlich für den Eigentümer (u), die Gruppe (g), allen anderen (o) oder alle drei (a für all). Ein Pluszeichen bedeutet, das Recht wird hinzugefügt, ein Minuszeichen bedeutet, das Recht wird entfernt. Die Rechte selbst werden durch r, w oder x gekennzeichnet. Für das SUID-, SGID- und SVTX-Recht mit s oder t. Mit einem = wird ein Recht komplett neu gesetzt.

Soll für ein Unterverzeichnis inklusiv der darin vorhanden Dateien das Zugriffsrecht verändert werden, erreicht man das durch die Option **-R** (rekursiv).

Man kann die Zugriffsrechte auch durch einen Zahlencode (Oktalzahl 0-7) setzen. Dabei werden den Buchstaben folgende Werte zugeordnet:

r=4, w=2, x=1, nichts = 0

Für das SUID-, SGID- und SVTX-Recht wird im numerischen Modus eine vierte Ziffer verwendet, die vor die Ziffern für den Eigentümer, die Gruppe oder der Klasse der „anderen“ gesetzt wird.

Das SUID-Recht hat den Wert 4, das SGID-Recht den Wert 2 und das SVTX-Recht den Wert 1.

Den richtigen Zahlencode erhält man durch Addieren der Werte der Einzelrechte die jede Benutzergruppe bekommen soll. Der Zahlenwert 644 ergibt folgende Zugriffsrechte:

-rw-r--r--.

Einige Beispiele:

Die Datei *MyScript* besitzt folgende Rechte:

```
-rwxr--r--   1  il001 labor      2703 Jul 13:18 MyScript
```

Mit dem Kommando: chmod u-x MyScript oder chmod 644 MyScript nimmt der Eigentümer sich selbst das Recht, die Datei ausführen zu lassen.

```
advm1> ls –l MyScript
–rw-r--r--   1  il001 labor      2703 Jul 13:18 MyScript
```

Mit dem Kommando chmod u+s MyScript oder chmod 4555 MyScript setzt man das SUID-Recht für die Datei *MyScript*.

```
advm1> ls –l MyScript
–rwsr-xr-x   1  il001 labor      2703 Jul 13:18 MyScript
```

Mit dem Kommando chmod g+s MyScript oder chmod 2750 MyScript setzt man z.B. das SGID-Recht für die Datei *MyScript*.

```
advm1> ls –l MyScript
–rwxr-s---   1  il001 labor      2703 Jul 13:18 MyScript
```

In dem letzten Beispiel wird für das HTML-Verzeich *public_html* das Zugriffsrecht mit dem Zahlencode 701 gesetzt. Der Browser sucht nach den Dateien *index.html* oder *Welcome.html*.

```
advm1> chmod 701 public_html
advm1> ls –l public_html
–rwx-----x   1  il001 labor      512 Jul 13:20 public_html
```

3.4.6 Zugriffsrechte vorgeben

Bei jeder Erstellung einer Datei werden automatisch vom System voreingestellte Zugriffsrechte gesetzt. Diese Voreinstellungen können bei Sitzungsbeginn auf die eigenen Bedürfnisse geändert werden und zwar mit dem Befehl:

umask Oktalzahl

Die Oktalzahl muss der invertierte Wert der beim **chmod** angegebenen Oktalzahl sein. Die Bits, die in der Maske angegeben sind, werden beim Anlegen der Datei gerade **nicht** gesetzt. umask bewirkt also nicht das Setzen der Zugriffsberechtigungen, sondern das Nichtsetzen. Ein Beispiel:

umask 077

Jede von diesem Zeitpunkt an erzeugte Datei wird mit read, write- und execute-Berechtigung für den Eigentümer erzeugt, andere Benutzer haben überhaupt keine Zugriffsrechte.

Bei den meisten Unix-Systemen ist noch eine Besonderheit bei der Verwendung des umask- Kommandos zu beachten. Ein x-Bit wird nur dann gesetzt, wenn die neu erstellte Datei ein Verzeichnis ist oder von einem Compiler- oder Linklauf erzeugt wurde. Z.B. haben Dateien, die von einem Editor nach

umask 022

erzeugt werden, folgende Zugriffsrechte: **`–rwxr--r--`** und nicht, wie vielleicht vermutet, **`–rwxr-xr-x.`**

Existierende Dateien werden bei einer Änderung von **umask** nicht beeinflusst. Die Rechte müssen explizit mit **chmod** geändert werden.
Sollte die Voreinstellung dauerhaft geändert werden, dann ist es sinnvoll, diese in die Konfigurationsdatei ***.tcshrc*** einzutragen. Dann wird bei jedem Aufruf der tcsh-Shell diese Voreinstellung gesetzt und bleibt für die Dauer der Arbeiten aktiv.

3.4.7 Eingabe über ein Terminal in eine Datei

Mit dem **cat-Kommando** (con**cat**enate) kann man Dateien erzeugen, indem man den Inhalt über ein Terminal eingibt. Dies geschieht folgendermaßen:

cat > *dateiname*
Eingabetext; Der Text kann über mehrere Zeilen gehen.
<ctrl d>

Man schließt die Eingabe am Terminal mit ***<Return>*** und ein nachfolgendes ***<ctrl d>*** ab.

Der Nachteil dieser Eingabeform ist, man kann keine Tippfehler in vorhergehenden Zeilen korrigieren. Dafür muss man dann den Editor benutzen.

3.4.8 Ausgabe auf dem Terminal

Für die Ausgabe einer Datei auf dem Terminal verwendet man den Befehl:
cat -Optionen Dateiname

-e	Am Ende jeder Zeile wird ein $ angezeigt.
-t	Im Text gesetzte Tabulatoren werden als ^I angezeigt.
-v	Zeichen, die normalerweise am Bildschirm nicht darstellbar sind, werden als ^x oder ^? angezeigt.

Bei längeren Ausgaben auf dem Terminal geben es drei Möglichkeiten die Ausgabe anzuhalten:

Durch Drücken der Tasten **<ctrl s>**. Mit <**ctrl q>** hebt man den Stopp wieder auf.
Bei Verwendung einer Pipe (siehe 3.1) lautet der Befehl dann folgendermaßen:

cat *Dateiname* **|** ***more***

Die Ausgabe erfolgt wie unter ***more*** *Dateiname* angegeben.

Durch den Befehl

more *Dateiname*

wird eine Bildschirmseite ausgegeben und in der letzten Zeile steht invers geschrieben **--More--(34%)**. Nach Drücken der **<Return>**-Taste wird die nächste Zeile ausgegeben und hinter More eine neue Prozentangabe, wie viel Prozent der Datei bis jetzt ausgegeben sind. Nach

Drücken der **<Blank>**-Taste wird eine neue Bildschirmseite einschließlich einer neuen Prozentangabe ausgegeben.

3.4.9 Kopieren von Dateien

Mit dem Kommando ***cp*** (**c**o**p**y) kann man eine oder mehrere Dateien in eine neue Datei kopieren.

cp *Quelle Ziel* oder ***cp*** *Datei1 Datei2 ... Ziel*

Falls die Datei *ziel* schon existiert, dann ist der alte Dateiinhalt verloren!!

3.4.10 Löschen von Dateien

Mit dem Kommando ***rm*** (**re****m**ove) kann man eine oder bei Verwendung eines Sternoperators (*) alle Dateien in dem aktuellen Verzeichnis löschen.

rm *Datei* oder ***rm*** * oder ***rm*** *Name**

Befinden sich Unterverzeichnisse in einem Verzeichnis, so kann man mit folgendem Befehl alle Dateien eines Unterverzeichnisses (! sowie eventuell weitere vorhandene Unterverzeichnisse und ihre Dateien) und den Verzeichniseintrag selbst löschen:

rm –rf* oder ***rm –rf*** *Dirname*

Sollten Sonderzeichen im Dateinamen sein (z.B. Leerzeichen, !, (,), -,), so gibt es folgende Möglichkeiten:

- Der Dateiname beginnt mit einem Minuszeichen (-);
  Beispiel: Die Datei *-ab* wird entfernt durch den Befehl

  rm - *--ab*

- Alle anderen Problemfälle kann man **interaktiv** lösen. Man reduziert durch Auswahl eines geeigneten Musters die Anzahl der zu bearbeitenden Dateien, so dass man durch wenige Eingaben (y oder n) die gewünschten Dateien löschen kann.

  rm -i **ab*

  Der angegebene Parameter **-i** (interaktiv) fordert den Benutzer vor jedem Löschen einer Datei zur Bestätigung auf, ***ab** zeigt alle Dateien an, deren Namen mit ab endet.

Eine mit *rm* gelöschte Datei kann nicht wiederhergestellt werden.

3.4.11 Umbenennen einer Datei

Mit dem Kommando ***mv*** (**m**o**v**e) kann man eine Datei umbenennen.

mv Alter_name Neuer_name

Hierbei muss man darauf achten, dass die Datei *Neuer_name* noch nicht existiert, da sie sonst gelöscht wird.

3.4.12 Anlegen von leeren Dateien

Einige Anwendungen erwarten manchmal, dass eine bestimmte Datei bestehen muss. Eine einfache Methode, solche Dateien anzulegen, ist die Verwendung des Kommandos ***touch***.

touch *dateiname*

Beim Aufruf des Kommandos wird eine Leerdatei angelegt, sofern noch keine Datei mit diesem Namen besteht. Es können auch mehrere Dateinamen hinter dem Kommandonamen angegeben werden.

4 Shell

Die **Shell** ist eine Art Benutzerschnittstelle zwischen dem Betriebssystem und dem Anwender und kann interaktiv Kommandos ausführen.
Die Shell besitzt eine Reihe nützlicher Eigenschaften wie z.B.:

- Interaktive Kommandointerpretation.
- Hintergrundverarbeitung (Batch).
- Ein-/ Ausgabeumlenkung
- Pipelines
- Wildcardauflösung.

Während einer Sitzung am Rechner kommuniziert der Benutzer normalerweise nicht direkt mit dem Betriebssystem, sondern mit einem Programm, welches seine Kommandos liest, analysiert und dann entweder selbst ausführt oder an andere Programme weitergibt. Dieses Programm wird deshalb als Kommandointerpreter bezeichnet und trägt den Namen **Shell**, weil sie wie eine Schale um den Betriebssystemkern liegt. Unix Shell-Befehle können sowohl als interne als auch als externe Befehle ausgeführt werden. Die internen Befehle (internal commands) sind fest eingebaute (built-in) Kommandos der entsprechenden Shell. Bei den externen Kommandos wird ein neuer Prozess zur Befehlsausführung gestartet. Die Ausführung eines Kommandos erfolgt nach folgender Regel:

Ist der Aufruf ein interner Befehl? Ja: Wird direkt von der Shell ausgeführt.
Nein: Besitzt der Aufruf eine absolute Pfadangabe? Ja: ausführen.
Nein: Benutze die Pfade, die in der Umgebungsvariablen PATH festgelegt sind und führe den Befehl aus, wenn er dort gefunden wird. Sonst erfolgt eine Fehlermeldung.

4.1 Arbeitsumgebungen von Benutzern

Es geben fünf verschiedene Shells, wie in Kapitel 1.2 angedeutet, mit denen man sich die Arbeitsumgebung einstellen kann. Diese Shells sind frei wählbar. Die Shell führt mindestens eine Startup-Datei, manchmal sogar zwei (siehe nachfolgende Tabelle) aus. In Gummersbach wird standardmä-

ßig die *tcsh-Shell* zur Verfügung gestellt. Nachfolgend eine kurze Übersicht über die Eigenschaften der verschiedenen Shells.

4.1.1 Startup-Dateien

Die Shell kümmert sich um die Arbeitsumgebung. Zur Arbeitsumgebung eines Benutzers gehören ein Home-Directory, eine Login-Shell und unter anderem eine Reihe von Dateien, die wichtige Systemvariablen oder andere Besonderheiten festlegen. Sie setzt z.B. das Promptzeichen, teilt einem mit, ob eine Mail vorliegt, führt beim Einloggen die erste Startup-Datei *.tcshrc* und anschließend die zweite Startup-Datei *.login* aus. Nachfolgend eine Tabelle, welche Shell welche Startup-Datei(en) ausführt:

Shell	sh	csh	tcsh	ksh
1. Startup-Datei	.profile	.cshrc	.tcshrc	.profile
2. Startup-Datei		.login	.login	.kshrc

4.1.1.1 .cshrc oder .tcshrc

Die Datei ***.cshrc*** (C shell restart commands) oder ***.tcshrc*** wird bei jedem Subprozeß ausgeführt. Darum sollten in dieser Datei Kommandos enthalten sein, die nur für die jeweilige C-Shell, in welcher der Subprozeß läuft, gelten sollen.

Übliche Eintragungen:

Alias Substitutionen
Definition des Prompts
Path-Variable
History-Variable
Setzen von Umgebungsvariablen
Setzen von vordefinierten Variablen
Alle Variablen, die in Sub-Shells exportiert werden sollen

Beispiel:

```
# .tcshrc-Beispiel
alias h history
set correct=all
set autolist
set history=25
set autologout=60
set prompt="%B%U%m %/ %b%u%# "
setenv TERM vt300
alias lt 'ls –alt | more'
```

4.1.1.2 .login

Diese Datei wird nur beim Einloggen des Benutzers bearbeitet, wenn der Benutzer die C-Shell oder tcsh-Shell als Login-Shell in der Datei ***/etc/passwd*** stehen hat. In dieser Datei ist es sinnvoll nur solche Kommandos zu schreiben, die für die ganze Sitzung von Bedeutung sind. Dass sind z. B. Terminaleinstellungen, Definition von Environmentvariablen, Start eines Windowmanagers oder Start einer Applikation.

4.1.1.3 .profile

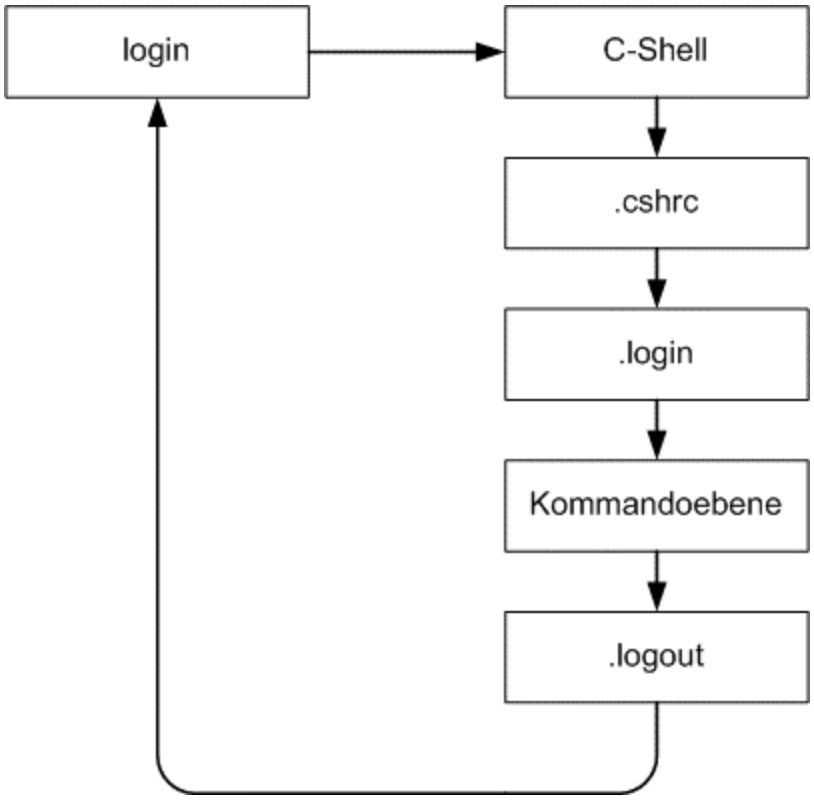
Diese Datei wird nach jedem Login abgearbeitet, wenn der Benutzer die Bourne-Shell ***sh*** oder die Korn-Shell ***ksh*** als Login-Shell in der Passwortdatei definiert hat.

4.1.2 Logout-Datei

Beim Ausloggen eines Benutzers, liest die csh-Shell und die tcsh-Shell - falls vorhanden - die Datei ***.logout*** und führt alle dort befindlichen Befehle aus. Dies sind z.B. Aufräumarbeiten, Starten eines Hintergrundprozesses oder Ausgabe eines Textes auf dem Bildschirm.

4.1.3 Arbeitsweise der csh-Shell bzw. tcsh-Shell

Beim Login eines Benutzers wird nach dem Start der Login-C-Shell zuerst nach der Datei ***$home/.cshrc*** oder ***$home/.tcshrc*** und anschließend nach der Datei ***$home/.login*** gesucht. In diesen Dateien werden die Funktionen und Dienste der C-Shell definiert. Bei Beendigung einer Sitzung kann die Nachbereitungsdatei der C-Shell ***$home/.logout*** ausgeführt werden. Siehe nachfolgende Abbildung.



4.2 Arbeiten mit der Shell

Die Ein-/Ausgabeumlenkung ist eine der herausragenden Eigenschaften der Shell. Unix verdankt dieser Fähigkeit zu einem großen Teil der Ruf, ein sehr flexibles Betriebssystem zu sein.

4.2.1 Die Standarddateien

Unix ordnet intern jedem Kanal eine Kanalnummer, den sogenannten Filedescriptor, zu. Anhand dieser Zahl identifiziert Unix eine Datei. Per Konvention ist für jedes Programm der Datenkanal mit der Nummer 0 als Standardeingabe geöffnet. Der Datenkanal 1 ist für die Standardausgabe und der Datenkanal 2 für die Standardfehlerausgabe geöffnet.

Gerätenamen	Gerät	Kanal	Symbol
Standardeingabe	stdin	0	<
Standardausgabe	stdout	1	>
Standardfehlerausgabe	stderr	2	>&

Die Shell ist nun in der Lage, für jeweils ein Kommando, das vom Benutzer eingegeben wird, diese Ein-/Ausgabekanäle so zu verändern, daß sie nicht mehr mit der Tastatur oder dem Bildschirm verbunden sind, sondern mit einer Datei. Man kann jedem der Umleitungssymbole die Kanalnummer voranstellen. In der **csh** bzw. **tcsh** sieht diese Regelung etwas anders aus. Die Standardfehlerausgabe kann man leider nicht alleine in eine Datei umlenken. Durch das Umlenksymbol >& wird immer die Standardausgabe und die Standardfehlerausgabe in die gleiche Datei umgelenkt.

Der allgemeine Kommandoaufruf lautet:

Kommando [Argumente] < Eingabedatei > Ausgabedatei

Beispiele: Siehe nachfolgende Punkte.

4.2.2 Umlenken der Eingabe (input redirection)

Durch das < -Zeichen wird die Standardeingabe einer Datei zugewiesen. Das Kommando

Kommando < Eingabedatei

liest die Eingabe für Kommando aus der Datei *Eingabedatei*.

Beispiel:

```
wc < datei
```

Das Kommando wc gibt die Anzahl der Zeichen, Wörter und Zeilen aus, die in der Standardeingabe (in diesem Fall die Datei *datei*) enthalten sind.

4.2.3 Umlenken der Ausgabe (output redirection)

Durch das > -Zeichen wird die Standardausgabe einer Datei zugewiesen. Das Kommando

Kommando > Ausgabedatei

schreibt die Ausgabe von Kommando in die Datei *Ausgabedatei*. **Eine eventuell existierende *Ausgabedatei* wird dabei überschrieben!!!** Um die Ausgabe an eine bestehende Datei **anzuhängen**, benutzt man das **>>** -Zeichen.

Einige Beispiele:

`cat datei`
Der Inhalt der Datei *datei* erscheint auf dem Bildschirm.

`cat datei1 > datei2`
Der Inhalt der Datei *datei1* wird in die neue Datei *datei2* geschrieben. Dieser Befehl ist gleichbedeutend mit *cp datei1 datei2* .

`cat /dev/null > datei`
Erzeuge die leere Datei *datei* . Dieser Befehl ist gleichbedeutend mit *touch datei* .

`ps –fu il001 >> datei`
Eine Liste aller Prozesse des Benutzers il001 wird an die **vorhandene** Datei *datei* angehängt.

Eine Umleitung der Standardausgabe und der Standardfehlerausgabe in eine Datei würde für die **csh- bzw. tcsh Shell** folgendermaßen aussehen: (nur beide gleichzeitig möglich)

`a.out >& Ausgabedatei`

In der **Bourne-Shell** könnte man die normale Ausgabe in eine Datei und die Fehlerausgabe in eine andere Datei schreiben. Siehe Beispiel:

`$ a.out > Ausgabedatei 2> Fehlerausgabedatei`

4.2.4 Pipelines

Für die direkte Verknüpfung des Standardausgabekanals eines Kommandos mit dem Standardeingabekanal eines anderen Programmes existiert die Pipe. Ein senkrechter Strich „|“ ist das „Pipe-Symbol“ zur Erzeugung solch einer Pipe. Man kann damit folgendes bewirken:

Eingabe -> Kommando1 -> Zwischenergebnis -> Kommando2 -> Ausgabe

Solch eine Verkettung, die beliebig viele Kommandos verbinden kann, nennt man eine Pipe. Somit kann man Filterprogramme schreiben, die durch Pipelines miteinander verbunden sind. Das Pipelinezeichen kann mehrfach in einer Eingabezeile vorkommen.

Beispiel:
`ls -al | more`
Dieses Kommando gibt eine Liste aller Dateien des aktuellen Verzeichnisses seitenweise aus.

4.2.5 Shell-Variablen

In Shell-Variablen kann man Texte für den späteren Gebrauch speichern. Bei Abfragen von Shell-Variablen wird der gespeicherte Text zurückgegeben. Der Datentyp der Shell-Variablen ist davon abhänig, in welchem Zusammenhang die Variable abgefragt wird (kontext-sensitiv). Shell – Variablen setzt man ein für:

- Speicherung benutzerspezifischer Konfigurationen der Shell.
- Übergabe von Kommandozeileninhalten an Skriptdateien.
- Temporäres Speichern von Texten während der Skriptverarbeitung.
- Setzen von Synonymen (Aliasfunktion) für eine komfortablere Arbeit mit der Shell.

Gewisse Variablennamen sind festen Zwecken zugeordnet und **sollen** daher nicht anderweitig verwendet werden (z.B. PATH). Die Sichtbarkeit von Shell-Variablen ist auf die erzeugende Shell begrenzt und damit für andere Prozesse nicht nutzbar. Nur durch ein exportieren in die Umgebungsvariablenliste (siehe Kapitel 4.2.6) können sie außerhalb genutzt werden.

Per Konvention sollten alle benutzerdefinierten Variablen klein geschrieben werden, also nur aus Kleinbuchstaben, Ziffern und dem Unterstrich bestehen. Dadurch kann man sie besser von den Systemvariablen unterscheiden.

Das Einrichten einer Variablen geschieht folgendermaßen:
`a=Text`

Damit ist die Shell-Variable a eingerichtet und mit dem Text *Text* vorgesetzt. Wichtig: Weder rechts noch links vom Gleichheitszeichen dürfen Leerzeichen stehen.
`b=`
bedeutet, die Shell-Variable existiert, sie hat aber keinen gültigen Wert.

Eine Shell-Variable kann in Ihrem Text auch Sonderzeichen, meistens Leerzeichen enthalten. Damit man Leerzeichen in einer Variablen unterbringen kann, muß man die Leerzeichen vor der Shell verstecken. Am einfachsten geschieht dies durch Anführungszeichen (“).

`a="Susi ist lieb"`

Alternativ kann man auch Hochkommata (’) oder auch den Backslash (\) verwenden. Wichtig ist, das Anführungszeichen und Hochkommata verschiedene Bedeutungen haben. Bei Anführungszeichen wird das Dollarzeichen und die Substitution ausgewertet während bei Hochkommata z. B. $b ausgegeben wird und nicht der Inhalt von b.

Nachfolgend eine Liste der wichtigsten Systemvariablen, die standardmäßig auch exportiert werden, damit sie an Kindprozesse weitervererbt werden können.

Variable	Funktion
HOME	Das Verzeichnis, das mit dem CD-Kommando (ohne Argumente) und nach dem Login erreicht wird.
PATH	Die Variable PATH gibt den Kommando-Suchpfad für die Shell an. Diese Variable enthält volle Pfadnamen zu den Dateiordnern, die durchsucht werden müssen. Ein Doppelpunkt (:) schließt jeweils einen Pfad ab. Die angegebenen Ordner werden in der Reihenfolge durchsucht, in der sie PATH angegeben sind.
TERM	Bezeichnung des angeschlossenen Terminaltyps.
USER	Benutzerkennung.
SHELL	Voller Pfadname des Shellinterpreters.

Nachfolgend eine Liste der wichtigsten Befehle für Shell-Variablen:

- set: Anzeige aller definierten und sichtbaren Shell-Variablen.
- echo: Inhalt einer Shell-Variablen ausgeben. (z.B. für a: echo $a).
- read: Einlesen eines Textes über die Tastatur bis die Return-Taste betätigt wird. (read a).
- <variablen name> = <variablen inhalt> Setzen mittels Kommandozeile (Setzen von a: a = Emma).
- $<variablen name> a wird von der Kommandozeileninterpretation durch den Inhalt ersetzt. (z.B. echo $a).
- unset <variablen name> Variable löschen. D.h. Leerinhalt erstellen. (z.B. unset a).

4.2.6 Umgebungsvariablen

Umgebungsvariablen werden für den Gebrauch in mehreren Shells benötigt, da sie zum Unterschied zu Shell-Variablen eine erhöhte Sichtbarkeit und Gültigkeit haben. Eine Umgebungsvariable wird erzeugt, in dem man zuerst eine lokale Shell-Variable anlegt und diese anschließend exportiert. Durch den Export wird die Shell-Variable in die Liste der Umgebungsvariablen des Prozesses hinzugefügt. Nun kann die Shell-Variable an Kindprozesse weitervererbt werden. Standardmäßig sind gewisse Shell-Variablen automatisch exportiert wie z.B. PATH.

Die Umgebungsvariablen kann man mit dem Kommando ***setenv*** *Variable Wert* ändern. Falls man Eintragungen auf Dauer ändern möchte, dann ist es sinnvoll, diese Änderungen in der Datei *.login* oder *.tcshrc* vorzunehmen.

Hinweis: Eine Umgebungsvariable wird immer nur auf nachfolgend gestartete Programme vererbt. Bereits laufende Programme erfahren keine Änderung, da bei jedem Programmstart für das

neue Programm eine Kopie der Umgebung angelegt wird. **Änderungen werden auch niemals an den Vaterprozeß weitergegeben**.

4.2.7 Die Alias Funktion

Die Alias Funktion ("Zweit-Namens-Vergabe") ermöglicht die Zusammenfassung von komplexen Kommandos mit mehreren Optionen und Argumenten oder sogar Kommandofolgen zu einem kurzen, prägnanten Kürzel. Durch Eingabe des Kürzels wird dieses durch die entsprechenden Kommandos, die hinter dem Kürzel stehen, ersetzt. Damit kann die Arbeitsumgebung flexibel angepaßt und häufig verwendete Kommandos sehr schnell aufgerufen werden. Einige Beispieleingaben:

```
alias h history
alias l ls –al
alias ll 'ls –al | more'
```

Man kann diese Anweisungen auch in die 1. Startup-Datei *.cshrc* oder *.tcshrc* eintragen, dann werden diese Eingaben automatisch ausgeführt.

4.3 Einige wichtige Unix-Kommandos

Nachfolgend eine Liste einiger wichtiger und angenehmer Unix-Kommandos in alphabetischer Reihenfolge:

4.3.1 bg – Programm im Hintergrund laufen lassen

Das Kommando ***bg*** *Job* stellt den angegebenen Job in den Hintergrund, wo er dann weiterarbeitet. Wird keine Job-Nummer angegeben, dann wird das gerade suspendierte Programm in den Hintergrund gestellt. Ein laufendes Programm kann man durch die Eingabe von ***<Strg z>*** oder ***<Ctrl z>*** suspendieren. Sinnvoll ist dies nur bei Programmen, die keine Terminaleingabe erwarten. Wenn ein im Hintergrund laufendes Programm beendet werden soll, dann kann man dies auf drei verschiedenen Arten ausführen:

Mit dem Befehl ***jobs*** stellt man die Anzahl der im Hintergrund laufenden Programme fest und beendet anschließend mit ***kill %****jobnummer* oder ***kill –9 %****jobnummer* das entsprechende Programm.
Man gibt den Befehl ***fg*** ein und beendet anschließend mit ***<Ctrl c>***.
Mit dem Befehl ***ps*** die Prozess-Id feststellen und anschließend mit ***kill –9*** *Prozess-ID* das Programm beenden.

4.3.2 clear – Bildschirm löschen

Das Kommando ***clear*** löscht den Bildschirm und der Cursur steht oben links.

4.3.3 cut – Spaltenweise Extraktion von Texten

Mit ***cut*** lassen sich spaltenweise Texte aus Textzeilen extrahieren.

In der nachfolgenden Tabelle sind einige Parameter aufgelistet.

-b	byteweise
-c	zeichenweise
-f	felderweise
-d	Trennzeichen zwichen den Feldern

Beispiel: **`cut –c 10-20 dateiname`**

Das 10. bis 20. Zeichen jeder Zeile wird ausgegeben.

4.3.4 date – Ausgabe von Datum und Uhrzeit

Das Kommando ***date*** zeigt das aktuelle Datum und die aktuelle Uhrzeit an.

In der nachfolgenden Tabelle sind einige Parameter aufgelistet, die nützliche Teilbereiche des Datums darstellen.

+%j	Tag des Jahres. (001 – 366)
+%d	Tag des Monats. (01 – 31)
+%H	Stunde des Tages. (00 – 23)
+%m	Monat. (01 – 12)
+%M	Minute (00 – 59)

4.3.5 df – Informationen über das Dateisystem

Der Befehl ***df*** zeigt Informationen über den Gesamtspeicherplatz und den verfügbaren Speicherplatz eines Dateisystems an. Auf den IBM-Rechnern erfolgt die Ausgabe in Blöcken a 512 Byte. Mit dem Befehl ***df -k*** erfolgt die Ausgabe in Blöcken a 1024 Byte. Mit dem Befehl ***df -kI*** wird die Ausgabe der belegten I-Nodes unterdrückt.

4.3.6 diff – Vergleicht zwei Textdateien

Der Befehl ***diff*** *-optionen datei1 datei2* vergleicht zwei Textdateien und gibt deren unterschiedlichen Zeileninhalte aus.

-w	Ignoriert Tabulatoren und Leerzeichen.
-i	Ignoriert den Unterschied zwischen Groß- und Kleinschreibung.

4.3.7 du - Plattenauslastung

Der Befehl ***du*** zeigt die Blockanzahl aller für Datei und Verzeichnis angegebenen Dateien und Verzeichnisse an sowie rekursiv aller Verzeichnisse im angegebenen Verzeichnis. Die Zählung der Blöcke erfolgt in Einheiten a 512 Byte. Mit dem Befehl ***du -k*** erfolgt die Zählung in Einheiten a 1024 Byte.

4.3.8 env – Umgebungswerte anzeigen

Das Kommando ***env*** zeigt die aktuellen Umgebungswerte an.

4.3.9 expr – Einfache Rechenoperationen ausführen

Mit dem Kommando ***expr*** kann man einfache Rechenoperationen auf der Shell ausführen. Die Ausgabe erfolgt über die Standardausgabe. Meistens weist man das Ergebnis durch eine Kommandosubstitution einer Variablen zu.

expr *arg1 operator arg2 [operator arg3 …]*

Beim Aufruf dieses Befehls können Ausdrücke, Zeichenketten-, arithmetische oder Vergleichsausdrücke als Parameter übergeben werden.

```
expr 7 + 3
```

oder im Scriptfile

```
ergebnis = `expr 12 + 3`
echo $ergebnis
```

4.3.10 fg – Programm in den Vordergrund zurückholen

Das Kommando ***fg*** *Jobnummer* holt den angegebenen Job in den Vordergrund. Wird keine Jobnummer angegeben, dann wird der Job, der als letztes in den Hintergrund gestellt wurde, in den Vordergrund zurückgeholt.

4.3.11 find – Dateien mit bestimmten Eigenschaften suchen

Der Befehl ***find*** *Startverzeichnis Filterfunktion* durchsucht alle angegebenen Pfade und deren Unterverzeichnisse nach Dateien, die die angegebenen Kriterien erfüllen. Die Kriterien werden dabei in Form von Filterfunktionen angegeben. Bei jeder Datei innerhalb des Suchpfades wird überprüft, ob die durch den Filter gestellte Bedingung erfüllt ist. Wenn ja, wird sie an den nächsten Filter übergeben. Wenn nein, fällt sie aus der Überprüfung heraus. Einige der Filterfunktionen haben besondere Effekte wie z.B. das Drucken des Dateinamens (-print) oder das Ausführen eines Befehls (-exec, meist mit dem Dateinamen als Argument).

Bei der Anwendung des **find**-Befehls ist **größte Sorgfalt** geboten. Unbedacht aufgerufen erzeugt **find** eine außerordentlich hohe Netz- und Plattenauslastung. Aus diesem Grund sollte man **niemals** einen **find**-Befehl von einem Verzeichnis aus starten, das weit oben in der Verzeichnishierarchie steht wie etwa **/** oder **/home**. In diesem Fall würde sich *find* durch sämtliche Verzeichnisse unterhalb dieses weit oben angesiedelten Verzeichnisses durcharbeiten, um alle Dateien in allen Unterverzeichnissen auf die Kriterien der Filterfunktion hin zu überprüfen.

-name Dateiname	Sucht die Datei *Dateiname*.
-print	Gibt den kompletten Pfadnamen der Datei, beginnend mit dem zu durchsuchenden Verzeichnis, auf der Standardausgabe aus.

-exec Befehl	Der angegebene Befehl wird ausgeführt. Die Stelle, an der in der Befehlszeile der Dateiname stehen soll, muss mit {} markiert werden. Jeder Befehl muss mit einem Semikolon abgeschlossen werden. **Vorsicht:** Da das Semikolon für die Shell eine Sonderfunktion hat, muss es als '; ' eingegeben werden.

Beispiel 1: Zeige alle Dateien an, die *auf1.c* heißen und sich unterhalb des Verzeichnisses */usr/local* befinden.

```
find /usr/local –name auf1.c –print
```

Beispiel 2: Zeige alle Dateien an, die mit *auf1* beginnen und sich unterhalb des Verzeichnisses */usr/local* befinden.

```
find /usr/local –name ”auf1*“ –print
```

Beispiel 3: Lösche alle *core* -Dateien im eigenen Homeverzeichnis mit allen Unterverzeichnissen.

```
find $HOME –name core –exec rm '{}' ';'
```

4.3.12 finger – Benutzerinformationen

finger zeigt an, welche Benutzer gerade eingeloggt sind oder Informationen über Benutzer.

```
advm1> finger
gesper   Norbert Gesper    pts/13     Wed 11:17
wi123    Vorname Nachname  pts/7      Wed 09:12
```

Wenn man Informationen über sich selbst oder von anderen wissen möchte, dann kann man dies durch Angabe der Userid, Vorname oder auch Nachname nach ***finger*** erreichen.

```
advm1> finger il001
Login name: il001                  In real life: Norbert Gesper
Directory: /home/il001               Shell: /usr/bin/tcsh
```

4.3.13 grep – Suchen einer Zeichenkette in Dateien

Das Kommando ***grep*** *-Optionen Suchstring Datei ...* sucht in allen angegebenen Dateien nach Zeilen, die die als Suchstring angegebene Zeichenfolge enthalten. Der Name der Datei sowie die gefundene Zeile werden ausgegeben. Gibt man keine zu durchsuchende Datei an, so liest **grep** von der Standardeingabe (nützlich bei Verwendung in Pipes). Enthält der Suchstring Leerzeichen oder Sonderzeichen, so muss man ihn in einfache Anführungszeichen setzen. Nachfolgend einige Optionen:

-i	Ignoriert Groß- und Kleinschreibung.
-l	Nur die Namen der Dateien, die den Suchstring enthalten, werden angezeigt. Die Ausgabe der entsprechenden Zeilen wird dagegen unterdrückt.
-v	Zeigt alle Zeilen an, die dem Suchmuster **nicht** entsprechen.
-w	Führt eine Wortsuche durch.
-x	Es wird nur nach Zeilen gesucht, die **genau** dem Suchstring entsprechen, also keinerlei sonstige Zeichen enthalten.

Beispiel 1: Liste alle Dateien auf die genau die Zeile *main()* enthalten.

```
grep –x "main()" .c*
```

Beispiel 2: Zeigt alle Zeilen an, die mit einem **d** beginnen. Läßt also nur Verzeichnisse durch.

```
ls –al | grep '^d'
```

Beispiel 3: Liste alle Prozesse des Users wi001 in seitenweiser Ausgabe auf.

```
ps –ef | grep wi001 | more
```

4.3.14 gzip und gunzip – Dateien komprimieren und dekomprimieren

Mit dem Programm ***gzip*** werden Dateien komprimiert und dadurch erheblich Plattenkapazität eingespart. Mit dem Programm ***gunzip*** werden Dateien die durch ***gzip*** komprimiert wurden wieder dekomprimiert. Der Aufruf lautet:

gzip *Dateiname*

Die neu erzeugte Datei bekommt den Namen *Dateiname****.gz***. Die ursprüngliche Datei wird gelöscht.

Mit dem Programm ***gunzip*** *Dateiname* kann man alle Dateien dekomprimieren deren Dateinamen mit folgenden Kürzeln enden: ***.gz, -gz, .z, -z, _z*** oder ***.Z***.

4.3.15 head – Zeigt die ersten Zeilen einer Datei an

Das Kommando ***head*** *-n Dateiname* gibt die ersten n Zeilen der Datei *Dateiname* auf dem Bildschirm aus. Die Voreinstellung für n ist 10. Will man z.B. die ersten 20 Zeilen ausgeben, dann lautet das Kommando:

head -n 20 Dateiname

4.3.16 jobs – Listet Hintergrundprozesse auf

Das Kommando ***jobs*** gibt Informationen über jeden im Hintergrund laufenden Prozess. Die Option **-l** listet zusätzlich zu den üblichen Informationen noch die Prozess-IDs auf.

4.3.17 kill – Prozesse beenden

Mit dem Kommando ***kill*** *-Signal Process-id* kann man einem Prozess ein Signal senden, um ihn z.B. abzubrechen, wenn er sich in einer Endlosschleife befindet.

Anmerkung: Die Prozess-Id eines Programms kann man mit dem Kommando ***ps*** feststellen. Nachfolgend einige Signale:

1	(SIGHUP) signalisiert das Ende der logischen Terminalverbindung (z.B. durch ein LOGOUT)
3	(SIGQUIT) signalisiert den Abbruch des Prozesses. Z.B. wird bei Betätigen der Interrupt-Taste <ctrl-c> einem im Vordergrund laufenden Prozess das Signal Nr. 3 gesendet, was zu einem Abbruch führt.
9	(SIGKILL) bricht den Prozess ab. Dieses Signal kann vom empfangenden Prozess nicht abgefangen werden.
15	(SIGTERM) Standardwert. Der Prozess erhält die Aufforderung, sich zu beenden. Es liegt jedoch an dem Prozess, ob er dies wirklich macht. Ein Editor kann z.B. noch eine geänderte Datei speichern oder temporäre Dateien entfernen.

Das Signal SIGKILL lässt dem empfangenden Prozess keine Gelegenheit notwendige Abschlußarbeiten durchzuführen. SIGKILL sollte deswegen nur dann benutzt werden, falls SIGHUP oder SIGTERM nicht erfolgreich sind.

Beispiel 1: Den Prozess mit der Prozeß-Id 1095 mit dem Standardsignal abbrechen.

```
kill 1095
```

Beispiel 2: Beendet den Prozess, deren Eigentümer der Benutzer mit der Prozeß-Id 201 ist.

```
kill -9 201
```

4.3.18 less – Dateien seitenweise anzeigen

Mit dem Programm ***less*** kann man eine Datei seitenweise am Bildschirm ausgeben. Der Aufruf lautet:

less *Dateiname*
Nachfolgend einige Eingabeparameter:

Leertaste	blättert eine Seite weiter.
Return	geht in der Datei eine Zeile weiter.
y	geht in der Datei eine Zeile zurück.
d	blättert eine halbe Bildschirmseite weiter.
u	blättert eine halbe Bildschirmseite zurück.
b	blättert eine Bildschirmseite zurück.
g	springt an den Anfang der Datei.
G	springt an das Ende der Datei.
/string	sucht nach der Zeile, in der die angegebene Zeichenfolge zum ersten Mal nach der aktuellen Position auftaucht. (Vorwärtssuche).
?string	sucht nach der Zeile, in der die angegebene Zeichenfolge zum ersten Mal vor der aktuellen Position auftaucht. (Rückwärtssuche).
r	schreibt den Inhalt des Bildschirms noch einmal (refresh). Das kann z.B. nützlich sein, wenn Systemmeldungen den Bildschirmaufbau zerstört haben.
q	beendet das Programm

4.3.19 more – Dateien seitenweise anzeigen

Mit dem Programm ***more*** kann man eine Datei seitenweise am Bildschirm ausgeben. Der Aufruf lautet:

more *Dateiname*

Nachfolgend einige Eingabeparameter:

Leertaste oder <Ctrl-f> oder f	blättert eine Seite weiter.
<Ctrl-b> oder b	geht eine Seite zurück.
Return	geht eine Zeile weiter.
3f	3 Zeilen weiter.
3b	3 Zeilen zurück.
1G oder g	springt zum Anfang.
G	springt an s Ende.
30G	Mache die 30. Zeile zur aktuellen Zeile.
/string	sucht nach der Zeile, in der die angegebene Zeichenfolge zum ersten Mal nach der aktuellen Position auftaucht. (Vorwärtssuche).
n	Suche nächstes Vorkommen von string
q	beendet das Programm

4.3.20 ps – Prozesse auflisten

Das Kommando ***ps*** *-Optionen* informiert über den Status aktiver Prozesse. Ohne Angabe von Optionen erhält man lediglich Informationen über die eigenen Prozesse, die vom selben Terminal wie das ***ps*** Kommando gestartet wurden. Zu beachten ist, dass in einem Window-System jedes Fenster wie ein Terminal behandelt wird.

Nachfolgend einige Optionen:

-e	Zeigt alle Prozesse bis auf Kernelprozesse an.
-f	Erzeugt eine ausführliche Liste der Prozesse.
-l	Generiert eine Liste mit allen Informationen.
-u	Zeigt alle Prozesse der zugehörigen Userid an.

Beispiel 1: Alle eigenen Prozesse anzeigen.

```
ps –u Userid
```

Beispiel 2: Alle Prozesse auf dem Rechner seitenweise anzeigen.

```
ps –ef | more
```

Dieses Kommando erzeugt z.B. folgende Teilausgabe:

```
   UID   PID  PPID  C     STIME   TTY    TIME COMMAND
  root     1     0  0    Jul 23     -   10:30 /etc/init
  root  8014     1  0    Jul 23     -    0:00 /usr/sbin/srcmstr
  root 11352  8014  0    Jul 23     -   11:02 /usr/sbin/inetd
gesper 13644 22906  3  07:50:13 pts/1    0:02 –tcsh
gesper 39504 13644 21  10:49:50 pts/1    0:00 ps -ef
```

Die Angaben bedeuten im Einzelnen:

UID	Username des Prozeßeigentümers.
PID	Prozess-Nummer des laufenen Prozesses.
PPID	Vater-Prozess-Nummer (**P**arent **P**rocess **ID**entification).
C	Scheduling-Parameter.
STIME	Start des Prozesses.
TTY	Name der Dialogstation, von der dieser Prozess gestartet wurde. (- keiner Dialogstation zugeordnet).
TIME	bisher vom Prozess verbrauchte CPU-Zeit.
COMMAND	Name des Kommandos, das diesen Prozess gestartet hat.

4.3.21 tail – Zeigt die letzten Zeilen einer Datei an.

Das Kommando ***tail*** *-n Dateiname* gibt eine Datei ab einem definierten Punkt auf die Standardausgabe aus.

Nachfolgend einige Optionen:

-f	Die Ausgabe wird permanent fortgesetzt, sobald an die Datei etwas angehängt wird.
-r	Revers, zeigt die Zeilen, beginnend am Dateiende, in umgekehrter Reihenfolge an.
-n	Anzahl der Zeilen, die angezeigt werden sollen. Standardwert ist 10.

Will man z.B. die letzten 20 Zeilen ausgeben, dann lautet das Kommando:
tail -n 20 Dateiname

4.3.22 tar – Erzeugung einer Archiv-Datei

Das Kommando ***tar*** *-Optionen Dateiliste* erzeugt eine Archivdatei oder restauriert Dateien einer Archivdatei.

Nachfolgend einige Optionen:

-c	Erstellt eine neue Archivdatei.
-f Archiv	Gibt an, dass das nächste Argument der Name des Archies ist, von dem gelesen oder auf das geschrieben werden soll. Wird anstelle eines Dateinamens ein Minuszeichen angegeben, so wird stdin oder stdout benutzt (je nachdem, was zum Key passt).
-r	Fügt Dateien einem bestehenden Archiv an.
-t	Listet die Namen der Dateien in einem Archiv auf.
-u	Fügt Dateien zu einem Archiv hinzu, wenn sie nicht bereits vorhanden sind oder wenn sie verändert wurden.
-v	Listet die Namen der gerade in Bearbeitung befindlichen Dateien.
-x	Extrahiert Dateien aus einem Archiv.

Beispiel 1: Lege ein Archiv Ihres C-Unterverzeichnisses in Ihrem Homedirectory an mit dem Archivnamen *C.tar*.

```
tar –cvf C.tar C
```

Beispiel 2: Den Inhalt der Archies *C.tar* wieder in seine ursprüngliche Form restaurieren.

```
tar –xvf C.tar
```

4.3.23 tr – Zeichenersetzung

Das Kommando ***tr*** ersetzt die Zeichen der Zeichenkette *string1* durch Zeichen der Zeichenkette *string2.* Der Aufruf lautet:

tr *[–Option] string1 [string2]*

-c	Ersetz Zeichen, die nicht in *string1* vorhanden sind.
-d	Entfernt die Zeichen in *string1* aus Zeichenfolge.
-s	Entfernt mehrfache Vorkommen der Zeichen in *string1*.

Mit dem Befehl **tr** kann man z.B. ein Zeichen gegen ein anderes Zeichen austauschen. Der folgende Aufruf ersetzt im Wort "Amme" alle "m" durch "n":

```
advm1> echo Amme | tr 'm' 'n'
Anne
```

Dabei sollte man beachten, dass in den Argumenten, die man übergibt, die Zeichen wie in Feldern angeordnet sind. Jedem Zeichen aus dem ersten Argument wird sein entsprechender Gegenpart aus dem zweiten Argument zugeordnet. So ersetzt **`tr 'abc' 'xyz'`** jeweils **`'a'`** durch **`'x'`, `'b'` durch `'y'` und `'c'`** durch **`'z'.`** Ist die zweite Zeichenkette kürzer als die erste, so füllt **tr** die Lücke mit dem letzten Zeichen aus der ersten Zeichenkette auf.

Mit der Option –d kann man z.B. einfach ein " Carriage Return" (Wagenrücklauf) aus einer Datei entfernen. Auf den verschiedenen Betriebssystemen sind die Zeilenumbrüche unterschiedlich definiert. In Unix ist es ein " Line Feed" \n , in DOS/Windows ist zusätzlich noch ein " Carriage Return" \r vorhanden. Mit dem Aufruf:

```
advm1> tr –d '\r' < windowstext > unixtext
```

kann man dies schnell erledigen. Zusammen mit der Option –c kann man Überflüssiges noch gezielter entfernen. Wenn man alles löschen möchte ohne die Groß- und Kleinbuchstaben, dann kann man das folgendermaßen erledigen. Mit der Option –c die Auswahl treffen und mit der Option –d die Zeichen löschen. Der Aufruf lautet:

```
advm1> tr –c –d 'A-Z a-z' < datei.txt
```

Mit der Option –s lassen sich Texte zusammenschrumpfen. So löscht das Kommando **`tr –s ' ' < datei.txt`** alle mehrfachen Leerzeichen in ein Leerzeichen.

4.3.24 uname – Ausgabe von Systeminformationen

Mit dem Befehl ***uname*** erhält man Systeminformationen.
In der nachfolgenden Tabelle sind einige nützliche Parameter aufgelistet.

-a Alle Informationen ausgeben
-n Rechnername
-p Prozessortyp
-s Name des Betriebssystems

Beispiel: **`advm1> uname -n`**

Mit diesem Beispiel wird der Rechnerame ausgegeben.

4.3.25 unzip – ZIP-Datei dekomprimieren

Mit dem Programm ***unzip*** werden Dateien die durch ein ZIP-Programm komprimiert wurden wieder dekomprimiert. Der Aufruf lautet:

unzip *Dateiname****.zip***

4.3.26 uptime

Das Kommando ***uptime*** gibt Informationen über den Rechner aus z.B.: wie lang läuft der Rechner, wie viele Benutzer sind eingeloggt und wie stark der Rechner ausgelastet ist.

```
advm1> uptime
12:12PM up 13 days 22:20 8 users, load average: 1.08,  1.28,  1.40
```

4.3.27 w - Welche Benutzer sind eingeloggt und führen welchen Befehl aus

w zeigt an, welche Benutzer an dem Rechner über welche Schnittstellen eingeloggt sind und welches Kommando gerade ausgeführt wird.

```
advm1> w
12:15PM up 13 days 22:23 8 users, load average: 1.80,  1.35,  1.39
User     tty        login@   idle   JCPU   PCPU what
ai010    pts/2      11:35AM     0      2      2 joe bruch.cc
wi123    pts/4       9:26AM    12      0      0 vi tfun.cc
```

wc – Zählt die Zeilen, Wörter und Buchstaben einer Datei

Mit dem Kommando ***wc*** zählt man Zeilen, Wörter und Buchstaben der Dateien, die in der Dateiliste angegeben sind und gibt das Ergebnis aus. Mit den Optionen **-l (Zeilen)**, **-w (Wörter)** und **-c (Buchstaben)** kann man auswählen, was gezählt werden soll. Wird keine Option angegeben, dann wird alles gezählt. Der allgemeine Aufruf lautet:

wc *-Option Datei1 Datei_n*

Einige Beispiele:

```
advm1> wc datei.cc
   32   236   1613 datei.cc

advm1> wc datei.cc sort.cc
   32   236   1613 datei.cc
```

```
   57   460   3114 sort.cc
   89   696   4727 total
```

4.3.28 whatis – Kurzinformation für ein Unixkommando

Das Kommando ***whatis*** *Kommando* liefert eine Kurzbeschreibung für ein Kommando. Die Angaben in Klammern (z.B. passwd(4) geben das Kapitel des Manuals an, in welchem sich der Eintrag befindet.

```
advm2> whatis passwd
passwd(1)       - Changes a user' password.
Passwd(4)       - Contains basic userattributes.
```

4.3.29 which – Programme auffinden

Das Kommando ***which*** *Dateiname* zeigt den absoluten Pfadnamen der Datei an. ***which*** sucht aber nur in den Verzeichnissen nach der Datei, die in der Umgebungsvariablen PATH angegeben sind. Einige Beispiele:

```
advm1> which xlC
/usr/bin/xlC

advm1> which prog
./prog
```

4.3.30 who – Welche Benutzer sind eingeloggt

who zeigt an, welche Benutzer an dem Rechner von woher eingeloggt sind.

```
advm1> who
ai010    pts/13     Jul 17 09:35   (advt2.GM.FH-Koel)
wi123    pts/7      Jul 17 09:45   (advm2)
ai056    pts/4      Jul 17 09:53   (berg-216.berg.ne)
```

5 Programmentwicklung

5.1 Aufruf von Compilern

Auf den IBM-Rechnern stehen folgende Compiler[3] zur Verfügung:

cc	C-Compiler (IBM)
c99	C-Compiler C99 Standard (IBM)
xlC	C++ Compiler (IBM)

[3] Übersetzer

gcc	GNU C-Compiler
g++	GNU C++ Compiler
javac	Java-Compiler (Sun)

5.1.1 C-Compiler

Der Aufruf des C-Compilers erfolgt mit ***cc*** oder ***c99***. Dateien, deren Namen mit ***.c*** enden, werden als C-Sourcedateien betrachtet und compiliert. Der C-Compiler produziert Objektdateien, die den gleichen Namen, jedoch die Endung ***.o*** besitzen. Wird nur ein einziges C-Modul übersetzt und anschließend automatisch gelinkt, so wird die Objektdatei wieder gelöscht. Die Ausgabedatei enthält das ablauffähige Programm und bekommt standardmäßig den Namen ***a.out***. Der Aufruf erfolgt folgendermaßen:

cc *-Optionen Datei . . .*

Nachfolgend eine Liste der wichtigsten Optionen:

-c	Die Datei wird compiliert, aber nicht gebunden. Die erzeugte Objektdatei bleibt erhalten und hat die Endung **.o**.
-o name	Die vom Linker erzeugte ausführbare Datei soll *name* und nicht *a.out* heißen.
-O	Der Optimierer für den C-Code soll aufgerufen werden.
-g	Der Compiler erzeugt zusätzliche Informationen in der Symboltabelle, die vom **Debugger** verwendet werden.
-s	(strip) Die Symboltabelle und sonstige unnötige Informationen werden aus der Ausgabedatei entfernt. Nicht anwenden, wenn zusätzlich die Option –g angegeben wird.
-l bibliothek	Benutzt die angegebene Objektdatei.

Informationen können mit dem Kommando ***man cc*** auf dem Bildschirm angezeigt werden.

Werden dem Compiler Dateien übergeben, deren Endung nicht ***.c*** lautet, so gibt er diese an den Linker weiter. Dasselbe geschieht mit Optionen (z.B. -s), die er nicht kennt.
Wenn das Headerfile für die Mathebibliothek (**<math.h>**) verwendet werden soll, so muss dem Compiler als letzte Option ***-lm*** angegeben werden, damit er die Bibliothek finden kann.

Beispiel 1: Standardaufruf: Die Quelldatei *test1.c* wird übersetzt und automatisch gelinkt. Das ablauffähige Programm steht in der Datei *test1*.

```
cc –o test1 test1.c
```

Beispiel 2: Die Quelldatei *test1.c* wird übersetzt und automatisch gelinkt (wobei durch die -s Option die Symboltabelle gestrippt wird, und durch die Option -l die Mathebibliothek *m* mit eingebunden wird), und das ablauffähige Programm erhält den Namen *test1*.

```
cc –s –o test1 test1.c -lm
```

5.1.2 C++ Compiler

Der Compiler für die Programmiersprache C++ heißt bei IBM ***xlC***. Die in C++ geschriebenen Programme müssen die Endung ***.cc*** oder ***.C*** besitzen, sonst wird nur der 'normale' C-Compiler aufgerufen. Die Beschreibung der Optionen siehe Kapitel 5.1.1 C-Compiler.
Der Aufruf lautet:

xlC *-Optionen Datei . . .*

Beispiel: Standardaufruf: Die Quelldatei *test1.C* wird übersetzt und automatisch gelinkt. Das ablauffähige Programm steht in der Datei test1.

```
xlC –o test1 test1.C
```

5.1.3 GNU C-Compiler

Ein weiterer C-Compiler ist der GNU C-Compiler (Public-Domain-Software[4]) mit Erweiterungen gegenüber dem Standard-C-Compiler. Die Benutzung ist identisch mit dem C-Compiler von IBM. Siehe Kapitel 5.1.1 C-Compiler.

Der Aufruf lautet:

gcc -Optionen Datei . . .

Beispiel: Standardaufruf: Die Quelldatei test1.c wird übersetzt und automatisch gelinkt. Das ablauffähige Programm steht in der Datei test1.

```
gcc –o test1 test1.c
```

5.1.4 GNU C++ Compiler

Ein weiterer C++ Compiler ist der GNU C++ Compiler und heißt ***g++***. Die in C++ geschriebenen Programme müssen die Endung ***.cc*** oder ***.C*** besitzen, sonst wird nur der 'normale' GNU C-Compiler aufgerufen. Die Beschreibung der Optionen siehe Kapitel 5.1.1 C-Compiler.

Der Aufruf lautet:

g++ *-Optionen Datei . . .*

Beispiel: Standardaufruf: Die Quelldatei *test1.C* wird übersetzt und automatisch gelinkt. Das ablauffähige Programm steht in der Datei *test1*.

```
g++ –o test1 test1.C
```

5.1.5 Java Compiler (Sun)

Für die Programmiersprache Java existiert ein entsprechender Compiler von Sun, der mit ***javac*** aufgerufen wird. Dateien, deren Namen mit ***.java*** enden, werden als Javaquelldateien betrachtet.

---

[4] Kostenlos im Internet verfügbar

Eine Beschreibung des Sprachumfanges sowie zugehöriger Programmierbibliotheken findet man mit Hilfe von einem Internetbrowser (seamonkey) unter folgender URL: **http://www2.gm.fh-koeln.de/**.

Der Aufruf des Compilers lautet:

javac *-Optionen Datei.java . . .*

Beispiel: Standardaufruf: Die Quelldatei *Test1.java* wird übersetzt und als Klassendatei (Bytecode) unter dem Namen *Test1.class* abgelegt.

```
javac Test1.java
```

Um das Programm ausführen zu können, muss es mit dem Java-Bytecode-Interpreter ***java*** geladen werden. Der Interpreter erwartet den Namen des auszuführenden Objektes, dies ist im Allgemeinen der Dateiname der erzeugten Class-Datei ohne die Extension .class. Die Fortsetzung des obigen Beispiels lautet dann:

```
java Test1
```

5.2 Entwicklungsumgebung Eclipse

Die freie Entwicklungsumgebung ***eclipse*** ist ein quelloffenes Programmierwerkzeug, ursprünglich für die Entwicklung von Java gedacht. Aufgrund seiner großen Erweiterbarkeit wird es inzwischen für viele Programmiersprachen eingesetzt.

Der Aufruf lautet:

eclipse

5.3 make

Das ***make***-Kommando beinhaltet das wichtigste (und mächtigste) Werkzeug zur Entwicklung von Programmsystemen. Es wurde zur automatischen Wartung von Programmen entworfen, deren Vollständigkeit und Korrektheit von Folgen anderer Dateien abhängt, die auf irgendeine Weise miteinander kombiniert werden müssen. Ein typisches, bei großen Programmen, die aus mehreren Modulen bestehen, auftretendes Problem ist, dass Änderungen nötig sind, sobald ein Quellcode-modul modifiziert wurde. Man hat dann zwei Möglichkeiten: Entweder man übersetzt das ganze System neu, was unter Umständen viel Zeit in Anspruch nehmen kann, oder man übersetzt nur die geänderten Module und geht dabei das Risiko ein, dass man irgendwelche Abhängigkeiten übersehen hat, was dann zu Inkonsistenzen führen kann. Mit dem ***make***-Kommando kann man diesen Vorgang automatisieren, dass nur die wirklich betroffenen Module kompiliert werden und zudem

garantiert ist, dass das neue System konsistent ist. Das ***make***-Kommando berücksichtigt folgende Punkte:

Die Aktionen hängen vom Datum ab.
Das *make*-Kommando selbst kennt einige Abhängigkeiten. Es weiß dass eine .o-Datei von einer zugehörigen .c, .C oder .cc-Datei über irgendeine Art von Übersetzung abhängt.
Außerdem kann es nötig sein, dass der Programmierer in die Lage versetzt wird, die Abhängigkeiten des Zielprogramms von diversen beteiligten Dateien festzusetzen.

Das ***make***-Kommando verarbeitet die Datei ***Makefile*** oder auch ***makefile***, die die Anweisungen für die Compilation enthalten. Wenn man als Dateinamen einen anderen Namen wie ***Makefile*** oder ***makefile*** geben möchte, dann sollte der Dateinamen mit ***.mak*** enden. Der einfache Aufruf lautet:

make

Der Aufruf mit einem anderen Dateinamen lautet:

make –f *Name****.mak***

Aufbau einer Regel:

<Symbol>:<Bedingung der Regel>
tab<Rumpf der Regel>

Aufbau einer make-Datei

```
# Kommentar
array: arrayFunctions.o arrayMain.o
<TAB>cc –o array arrayFunctions.o arrayMain.o

arrayMain.o: arrayMain.c arrayFunctions.h
<TAB>cc –c arrayMain.c

arrayFunctions.o: arrayFunctions.c arrayFunctions.h
<TAB>cc –c arrayFunctions.c

clean:
<TAB>rm –f arrayFunctions.o arrayMain.o

distclean: clean
<TAB>rm –f array
```

6 Netzwerkdienste

6.1 Das Internet

6.1.1 SSH – Gesicherter Login auf einen entfernten Rechner

Bei einem Login auf einem anderen Rechner mit ***telnet*** wird das Passwort unverschlüsselt über das Internet übertragen. Mit dem SSH-Protokoll werden das Passwort und die gesamte Verbindung verschlüsselt. So sind ein Mithören (Eavesdropping), das "Entführen von Verbindungen" (Connection Hijacking) und ähnliche Netzwerkangriffe nicht möglich.

Der allgemeine Aufruf für ssh lautet:

ssh Zielrechner -optionen

Gibt man als Option *-l Loginname* an, so wird für das Anmelden auf dem Zielrechner der entsprechende Loginname verwendet, sonst der Loginname, den man auf dem lokalen Rechner besitzt. Mit der Option -l verlangt der Zielrechner sofort das Passwort. Gelingt die Anmeldung, so wird auf dem Zielrechner die Standardshell des Benutzers gestartet. Ein Login auf dem Rechner *advm1* in Gummersbach könnte z.B. folgendermaßen lauten:

ssh advm1.gm.fh-koeln.de –l wi001 oder beim Rechenzentrum der Universität zu Köln

ssh dialog.rrz.uni-koeln.de –l userid