Arbeiten mit dem UNIX Werkzeugkasten

Unix, Linux, FreeBSD und Mac OS/X unter der Oberfläche

Holger Zuleger

HZNET – Training / Consulting / Netzwerke / Security

Inhalt

Vorwort

Die vorliegende Einführung in die Arbeitsweise mit dem Betriebssystem Unix wurde über einen Zeitraum von ca. 10 Jahren (1987–1997) erstellt und in der Erwachsenenbildung als begleitendes Unterichtsmaterial eingesetzt.

Eine erste Überarbeitung erfolgte etwas mehr als 10 Jahre danach (2009). Hier sind insbesondere Anpassungen am einführenden Kapitel vorgenommen worden und Hinweise zu Internetprogrammen sowie das Kapitel zur Textverarbeitung hinzugekommen. Diesem ist das Kapitel zum Drucken mangels Aktualität zum Opfer gefallen. Die Syntax der Kommandos wurde vereinzelt an die unterschiedlichen Betriebssytem Varianten, insbesondere an die unter Linux verwendete GNU Version angepasst. Nach wie vor sind die Kommandos jedoch sehr unvollständig in ihrer Optionsvielfalt und subjektiv in deren Auswahl wiedergegeben. Die Unterlagen sollen mitnichten einen Ersatz der Manualseiten darstellen.

Wiederum zehn Jahre später werden die Unterlagen unter die Creative Commons Lizenz (CC-BY-NC-SA)† gestellt und veröffentlicht. Nicht nachdem erneut einige Rechtschreibkorrekturen und Ergänzungen vorgenommen wurden.
Trotz mehrfacher Überarbeitung finden sich sicherlich noch immer eine Menge Rechtschreib- und evtl. auch inhaltliche Fehler. Die Literaturhinweise wurden nicht überarbeitet und sind als veraltet anzusehen geben sie doch den Stand von vor 30 Jahren wieder. Ein Buch möchte ich jedoch nach wie vor jedem empfehlen der sich mit dem Unix System beschäftigen und sein Grundprinzip verstehen möchte. Es handelt sich dabei um den „Unix Werkzeugkasten“ von Kernighan und Pike. Das Buch beschreibt zwar eine noch frühere Unix Version (Version 7) als die vorliegenden Unterlagen, zeigt allerdings in beeindruckender Weise das Zusammenspiel der „Tools“, die Unix dem erfahrenen Anwender bereit stellt.

Die vorliegenden Unterlagen sind in eben diesem Sinne erstellt worden und so kamen insbesondere der Editor *vi*(1), das Satzsystem *groff* (1) mit dem Makropaket `mm` (ergänzt um eigene Erweiterungen), sowie *gpic*(1) und *tbl*(1) bei der Erstellung der Unterlagen zum Einsatz. Der Index wurde durch selbstgeschriebene Shell Skripte erzeugt und *make*(1) dient als Kommandozentrale um all dies zu verbinden. Zur Nachbereitung wurde *ps2pdf* (1) und *pselect*(1) eingesetzt und die Versionskontrolle unterliegt seit neuestem *git*(1). So ist das vorliegende Dokument ein Beispiel für die Möglichkeiten von Unix, auch wenn das eine oder andere heutzutage als überholt und antiquarisch angesehen werden mag und insbesondere die Fixierung auf die Kommandozeile unter Umständen anachronistisch wirkt.

Wer jedoch heute Server unter Unix betreibt und diese Remote managen will, kommt mit *ssh*(1) und der Kommandozeile nach wie vor hervorragend aus. Und da fast alle Konfigurations- und Logdateien als Textdateien ausgelegt sind, ist der unter Unix angebotene Werkzeugkasten ein in vielerlei Hinsicht perfektes Hilfsmittel zur Konfiguration, Dokumentation, Sicherung und Wartung solcher Systeme.

Die Unterlagen sind dazu gedacht die grundlegende Idee des Unix Betriebssystems, und eine (hoffentlich) sinnvolle Auswahl seiner Werkzeuge vorzustellen und zu zeigen. Ob dies im Selbststudium gelingen mag ist zweifelhaft. Insofern sind die Unterlagen nach wie vor als begleitende Materialien zu einer praktischen Einweisung gedacht, und finden hoffentlich entprechende Verwendung.

Holger Zuleger im Juli 2017

---

† BY: Namensnennung; NC: Nicht Kommerziell; SA: Weitergabe unter gleichen Bedinungen
https://creativecommons.org/licenses/by-nc-sa/3.0/de/

1. Einführung

1.1 Entwicklung von UNIX

- Unix wurde 1969 in den Bell Laboratories (Tochter von AT&T) entwickelt.
- Diese erste Version wurde von Ken Thompson für die PDP 7 in Assembler geschrieben.
- Ziel war es, ein Betriebsytem zu schreiben, das besser als damalige batchorientierte Betriebsysteme für die Programmentwicklung geeignet ist.
- Die Entwickler haben von Anfang an mit dem Betriebsystem gearbeitet. Daher konnten Mängel frühzeitig erkannt werden.

1.1.1 Entwicklungsgeschichte der UNIX AT&T Linie

1969 Erste Version von UNIX für eine PDP 7 in Assembler geschrieben.

1970 Ken Thompson entwirft die maschinennahe Sprache B (abgeleitet von BCPL Basic Combined Programming Language) als Hilfsmittel zur Compilerentwicklung. Beide Sprachen kennen keine Datentypen (nur Maschinenworte) und sind nicht umfangreich genug.

1971 Aus der Sprache B entwickelt Denis Ritchie die Sprache C. Der Systemkern und die meisten Dienstprogramme werden in C implementiert (ca 10% sind noch in Assembler geschrieben).

1973 UNIX V5
Hauptsächlich für Ausbildungszwecke (Universitäten erhalten UNIX geschenkt).

1975 UNIX V6
Keine Unterstützung von Seiten des Herstellers, da AT&T sein UNIX nicht kommerziell vermarkten darf. (Hier begründet sich der vormals schlechte Ruf von UNIX.)

1977 Erste BSD UNIX Release

1979 UNIX V7
Ausgangsbasis für viele Portierungen (z.B. XENIX, Coherent, Minix).

1981 UNIX System III

1983 UNIX System V
Ab jetzt kommerzielle Vermarktung. AT&T bietet Schulungen und Wartung für UNIX an.

1984 UNIX System V.2
Offziell von AT&T als zukünftiger UNIX−Standard proklamiert. Die Festlegung dieses Standards erfolgt in der SVID (UNIX System V Interface Definition).

— Interprozesskommunikation (Semaphore, Messages, Shared Memory)
— Paging System

1986 UNIX System V.3 System V Interface Definition Issue 2

— Remote File System (RFS)
— Streams
— Shared Libraries

1987 UNIX System V.3.1

— Internationalisierung (versch. Zeichensätze, Datums-, Zeitformate usw.)

1989 UNIX System V.4 (USL UNIX System Labratories)

— Neue Struktur des Dateisystems
— Versuch der Integration der diversen UNIX Ableger (insbesondere BSD)
— Memory mapped files

1991 Solaris 1.0 erscheint
Linus Torvald beginnt seine Arbeit an Linux

1991 UNIX SVR4.2 (System V Release 4.2)

1993 Verkauf der UNIX Rechte an Novell (UnixWare)
BSD 4.4 erscheint

1999 Unix wird 30!
Linux Kernel 2.2 erscheint

2001 Linux Kernel 2.4
Mac OS X 10.0

2003 Mac OS X 10.3

2009 Unix wird 40

2019 Unix wird 50

2021 Linux wird 30

1.1.2 Die BSD – Linie

- Ab UNIX V7 wird an der „University of California at Berkley“ unabhängig von AT&T eine UNIX Version weiterentwickelt, die unter dem Namen BSD−UNIX (Berkley System Distribution) Verbreitung findet.
- Mit Unterstützung des DoD (Department of Defense) werden in Berkley viele Bibliotheksfunktionen und Dienstprogramme entwickelt, die inzwischen fast alle unter AT&T UNIX implementiert sind.
  - — Paging System
  - — Fast File System
  - — termcap
  - — vi
  - — xdb
  - — Sockets
- Abgeschlossene Version ist BSD 4.4, die seit 1995 verfügbar ist.
- Die bekannteste Implementierung, die auf BSD−UNIX basiert, ist das SUN Operating System. SUN hat allerdings bei der Weiterentwicklung des SUN OS mehr und mehr auf Konformität zu SysV geachtet. Heutige Solaris Versionen sind eine Mischung aus BSD−, SysV−Unix und SUN spezifischen Erweiterungen (z.B. Zonenkonzept).
- Freie BSD Implementierungen
  - — NetBSD
  - — FreeBSD
  - — OpenBSD

1.1.3 Linux

- Freie Implementierung eines Unix ähnlichen Kernels durch Linus Torvald
- Ergänzt um GNU Werkzeuge hat man eine vollständige freie Unix Implementierung die häufig mehr zur Verfügung stellt als kommerzielle Implementierungen
- Verschiedene Hersteller bieten fertige Distributionen bestehend aus Kernel plus GNU Tools und andere freie Software an
  - SuSE, OpenSuSE
  - RedHat
  - Debian
  - Ubuntu

1.1.4 Apple Mac OS/X

- Seit der Version 10 (In römischen Ziffern `X`) setzt Apple als Unterbau seines Betriebsystems ebenfalls eine BSD-Unix Variante (Darwin) ein. Allerdings wird dies erst sichtbar, wenn man ein Terminalfenster öffnet und auf der Kommandozeile arbeitet. Dann stehen nahezu alle typischen (BSD)Unix Kommandos zur Verfügung.
- In einigen Punkten hat Apple jedoch eigene Anpassungen und Veränderungen vorgenommen:
  - Die graphische Benutzeroberfläche (Aqua) ist eine Apple spezifische Desktop Anwendung, die auf dem Fenstermanager Quartz aufsetzt der ebenfalls eine Eigenentwicklung ist. Bei allen anderen Unix Systemen kommen, aufbauend auf dem graphischen Fenstermanager X-Windows, andere und meist unterschiedliche Desktop Systeme zum Einsatz.
  - Die Nutzerverwaltung folgt nicht den Unix Standards (Die Datei `/etc/passwd` wird nur im Single User Modus genutzt).
  - Auch das eingesetzte Filesystem wird man auf anderen Unix-Systemen nicht finden. Es erlaubt die Verwendung von zusätzlichen, detailierteren Dateirechten, zu deren Verwendung Mac OS/X spezifische Kommandos notwendig sind.

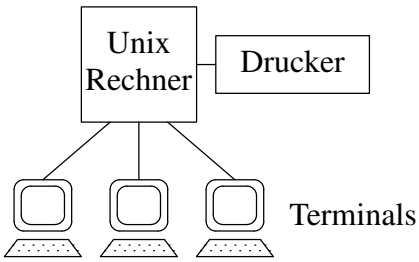
1.2 Eigenschaften von UNIX

- Interaktives System
- Multiuser Betriebsystem
- Multitasking Betriebsystem
- Hierarchisches Dateisystem
- Einheitliches Dateisystem (keine Unterscheidung von Laufwerken)
- Einheitliche Ein- und Ausgabe für Dateien, Geräte und Interprozesskommunikation.
- Hoher Grad an Portabilität durch Aufteilung in Kern und Schale bzw. durch Implementierung in einer höheren Programmiersprache.
- Graphische, netzwerkfähige Benutzeroberfläche (X11–Window System) mit unterschiedlichen Desktopmanagern (KDE, Gnome)

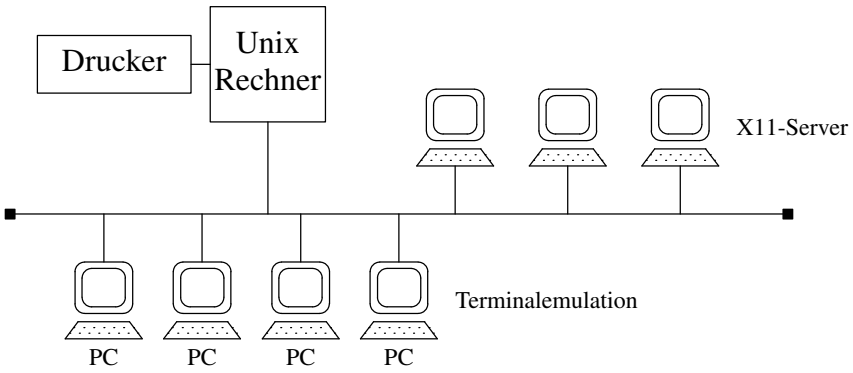
1.3 Die Architektur der UNIX–Umgebung

- Historisch erfolgt der Zugang zu einem UNIX–Rechner über Terminals. Terminals sind Computer mit Tastatur und Monitor, die jedoch über keine sonstige eigene Peripherie verfügen. Die Terminals sind über separate Leitungen (meist seriell RS232C

oder RS422) an den UNIX–Rechner angeschlossen.



- Der Unix–Rechner stellt allen Benutzern seine Hardware (d.h. Festplatten, Drucker, Streamer, Hauptspeicher, CPU) zu Verfügung.
- Heute sind UNIX–Rechner meist Workstations. Eine Workstation ist ein Einzelplatzsystem auf dem — im Falle von UNIX — ein Multi-Tasking und Multi-User Betriebsystem läuft.
- Arbeiten mehrere Nutzer auf einem Unix–Rechner erfolgt der Zugriff in der Regel über Datennetze (Internet).



- Hierfür wird auf den PCs eine Software eingesetzt die ein Terminal emuliert.
- Das bekannteste Programm dafür ist *telnet*(1). Dieses stellt jedoch eine unverschlüsselte Termoinalverbindung her.
- Heutzutage sollten Zugriffe auf entfernte Rechner ausschließlich verschlüsselt erfolgen. Dafür wird das Secure Shell Protokoll verwendet. Das Unix-Kommando dazu lautet *ssh*(1). Unter Windows wird meist der ssh-client *putty* verwendet um eine SSH Verbindung zu einem Unix System aufzubauen.
- Für graphische Anwendungen muss auf dem Arbeitsplatzrechner eine sog. X11-Server Software laufen. Applikationen können dann auf einem zentralen System gestartet werden. Die graphische Ausgabe wird über das Datennetz zu dem X11-Server übertragen.

Im folgenden geht es jedoch primär um textbasierte Terminalsitzungen und das Arbeiten mit der Kommandozeile.
Daher werden wir entweder ein Terminalfenster öffnen (Workstation) oder über *ssh*(1) bzw. *telnet*(1) auf einen anderen Rechner (Server) zugreifen.

2. Nutzung des Betriebsystem

2.1 Zugang zum System

2.1.1 Anmelden

- Da UNIX ein Multiusersystem ist, muss eine Zugangskontrolle stattfinden.
- Nach Einschalten des Terminals meldet sich das System mit der Loginmeldung.
- Danach ist der vom Systemverwalter zugewiesene Benutzername einzugeben.
- Wenn für den Benutzer ein Passwort vergeben wurde, kommt die Aufforderung dieses einzugeben.
- Nach erfolgreichem Einloggen meldet sich das System mit dem Promptsymbol (Früher ein einfaches Dollarzeichen[1]; heute meist eine Kombination aus User- und Rechnername sowie des aktuellen Dateipfades), sonst erscheint die Meldung `Login incorrect`.

```
$  telnet 192.168.1.132
Trying 192.168.1.132...
Connected to 192.168.1.132 (192.168.1.132).
Escape character is '^]'.
Welcome to openSUSE 10.3 (i586) - Kernel 2.6.22.5-31-default (4).

xt5 login: ben101
Password:
Have a lot of fun...
ben101@xt5:~>
$
```

2.1.2 Abmelden

- Auf keinen Fall einfach das Terminal ausschalten! Bei unserer Art des Zugangs zum Unix–Rechner bekommt dieser keine Nachricht wenn der PC ausgeschaltet wird. Der Unix–Rechner „denkt“ also immer noch, dass Sie angemeldet sind und wundert sich warum Sie nicht mehr arbeiten.
- Das Abmelden erfolgt mit dem Kommando `exit` oder durch die Taste `<^D>`. Das ASCII–Zeichen ^D hat die Bedeutung End of Text (EOT).

2.1.3 Nutzung eines Terminalfensters

- Arbeitet man auf einer Workstation (z.B. auf einem MAC Rechner) meldet man sich auf dem System über die graphische Benutzeroberfläche an.
- Um auf der Kommandozeilenebene zu arbeiten startet man lediglich die Terminalapplikation in einem Fenster.
- Eine Anmeldung ist dann nicht noch einmal erforderlich.

1. In diesem Dokument wird in den Kommandobeispielen die Eingabeaufforderung ebenfalls lediglich durch ein Dollarzeichen dargestellt.

2.2 Erste Schritte im System

2.2.1 Änderung (oder Neuvergabe) des Passwortes

- Jeder Benutzer kann zu jeder Zeit sein Passwort ändern.
- Das Passwort wird verschlüsselt im System abgelegt.
- Niemand kann dieses Passwort lesen (auch der Systemverwalter nicht). Wer sein Passwort vergisst, muss den Systemverwalter bitten es zu löschen.
- Das Kommando zur Änderung des Passwortes lautet `passwd`
  Beispiel:

```
$  passwd
Setting password for user Benutzername
Old password: Altes Passwort eingeben
Last   successful password change for Benutzer: Tue Mar 29 17:09:18 1993
Last unsuccessful password change for Benutzer: Tue Mar  1 10:01:20 1993

New password: neues Passwort eingeben
Re-enter password: Passwort erneut eingeben
```

2.2.2 Hinweise zur Tastaturbenutzung

- Die Eingabe von Kommandos erfolgt zeilenorientiert.
- Jede Eingabe muss durch <ENTER> abgeschlossen werden.
- Solange die Eingabe nicht abgeschlossen ist, kann die gesamte Eingabezeile verändert werden.
- Keine Funktionstasten oder z.B. <PageUp> oder <PageDown> benutzen. Auch die Cursortatsen funktionieren nicht unbedingt wie erwartet.
- Alle Korrekturen der Eingabe werden nur mit der <BackSpace>–Taste oder <§>–Taste durchgeführt. Heute unterstützen alle Kommandointerpreter auch das Editieren der Kommandozeile mit Cursortasten[2]. <BackSpace> (oder <^H>) löscht das zuletzt eingegebene Zeichen. Die §–Taste (oder <^U>) löscht die gesamte Eingabezeile.
- Jedes Kommando kann zu jedem Zeitpunkt mit der Taste <^C> abgebrochen werden.
  Beispiel:

```
$  ls -lR /      (lange laufendes Kommando)
            (während der Ausgabe <^C> drücken)
$           (System meldet sich mit Prompt)
```

- Bildschirmausgaben können im Terminalbetrieb in der Regel mit der Taste <^S> angehalten werden. Durch <^Q> läuft die Ausgabe weiter[3].
  Bei Datennetzverbindungen kann es bei der Übermittlung der Tastendrücke zu erheblichen zeitlichen Verzögerungen bei dem Erkennen der Tasten kommen.
- Alle Tasten lassen sich über das Kommando `stty` umdefinieren (Außer natürlich <Pause>).
  Die aktuelle Einstellung bekommt man durch `stty -a` angezeigt.

---

2. Genaueres zur Kommandozeile findet sic im Abschnitt Kommandointerpreter
3. Für die seitenweise Ausgabe gibt es spezielle Kommandos mit wesentlich größerem Funktionsumfang.

2.2.3 Allgemeine Kommandosyntax (nach SVID)

```
kommandoname [Optionliste] [--] [Parameterliste]
```

- Die in eckigen Klammern angegebenen Teile sind optional.
- Alle Teile müssen durch **white space** voneinander getrennt angegeben werden. Als *white space* wird eine Folge von <BLANK> oder <TAB> Zeichen bezeichnet.
- Der Kommandoname besteht aus beliebigen Zeichen. Je häufiger das Kommando benutzt wird, desto kürzer ist im allgemeinen der Kommandoname. UNIX unterscheidet Groß/Kleinschreibung. Alle UNIX Kommandos werden klein geschrieben.
- Optionen beginnen in der Regel mit einem Minuszeichen, gefolgt von einem Klein– oder Großbuchstaben. Dem Optionsbuchstaben kann — durch Leerzeichen getrennt — ein Parameter folgen. Einzelne Optionsbuchstaben können aneinander gehängt werden.
- Die spezielle Option `--` kann verwendet werden, um das Ende der Optionsliste anzuzeigen.
- Parameter stellen häufig Datei– oder Verzeichnisnamen dar. Sie sollten nicht mit einem Minuszeichen beginnen.

Achtung: Nicht alle Kommandos halten sich an diese Regeln. So werden manchmal Optionen mit einem Pluszeichen eingeleitet oder können nicht bzw. müssen zusammengefasst werden. Parameter von Optionen müssen meistens direkt an den Optionsbuchstaben gehängt werden.

Beispiele:

a. Kommando ohne Optionen und Parameter[4]:

pwd (print working directory)

Syntax:

pwd

Das Kommando zeigt den Pfadnamen des aktuellen Verzeichnisses an.

b. Kommando ohne Option mit optionalem Parameter

cd (change directory)

Syntax:

cd [*verzeichnis*]

Wechselt in das angegebene Verzeichnis.

Ist kein Verzeichnis angegeben, wird in das Homeverzeichnis gewechselt (steht in der Variablen `$HOME`).

4. Heutzutage hat selbst das sehr einfache Kommando `pwd` auf bereits Optionen.

c. Kommando mit Optionen und Parametern

ls (list files of directorys)

Syntax:

ls [**–abcCdfFgilmnopqrRstux**] [*name ...*]

`ls` ist ein Kommando zum Anzeigen eines Inhaltsverzeichnisses.

Die einfachste Form des Kommandos `ls` zeigt alle Dateinamen des aktuellen Verzeichnisses auf dem Bildschirm an.

Wird als Parameter ein Pfadname angegeben, werden alle Dateien des angegebenen Verzeichnisses ausgegeben.

Optionen beeinflussen die Darstellung der Liste:

`-a`	Alle Dateien anzeigen (auch die mit einem Punkt beginnenden)
`-b`	Nicht darstellbare Zeichen in einem Dateinamen werden als Oktalzahl dargestellt
`-C`	Dateinamen in mehreren Spalten ausgeben
`-d`	Ist als Parameter ein Verzeichnis angegeben, wird nicht eine Liste aller Dateien angezeigt sondern lediglich der Verzeichniseintrag.
`-F`	Schreibt hinter die Namen von Verzeichniseinträgen ein "/" und hinter ausführbare Programme ein "*". Nützlich in Kombination mit `-C`
`-l`	langes Ausgabeformat
`-r`	kehrt die Sortierreihenfolge um (Standard: nach Dateinamen aufsteigend)
`-R`	rekursive Ausgabe von Unterverzeichnissen
`-t`	Liste wird nach den Zeiteinträgen sortiert

Beispiele:

`ls /etc`	Gibt das Verzeichnis `/etc` auf dem Bildschirm aus.
`ls -a /etc`	Gibt alle Dateien des Verz. `/etc` aus.
`ls -a -l /etc`	Gibt eine lange Liste aller Dateien aus `/etc` aus.
`ls -al /etc`	s.o.
`ls -la /etc`	s.o.
`ls /etc /dev`	Zeigt alle Dateien aus `/etc` und `/dev` an.

Erläuterung der Ausgabe von `ls -l`.

```
$  ls -l
total 45
-rw-rw-r--  1 root  other    4712  Jun 22 09:18  anton
drwxrwxr-x  2 hoz   kaho       48  Mai  9 2007   beispiele
-rwxr-x--x  1 hoz   kaho    24516  Jun 19 12:13  clear
||         |  |   |        |      |     |      |     |
||         |  |   |        |      |     |      |     Dateiname
||         |  |   |        |      |     |     Uhrzeit oder Jahr
||         |  |   |        |      |     |  Datum d. letzten Änderung
||         |  |   |        |      |   Größe der Datei in Bytes
||         |  |   |      Gruppenname des Dateibesitzers
||         |  |  Benutzername des Dateibesitzers
||         |  Anzahl der Verweise auf die Datei
|\         /
|    Zugriffsrechte auf die Datei für:
|        Dateibesitzer         (rwx)
|        Gruppenmitglieder     (r-x)
|        Alle anderen Benutzer (--x)
|
Dateityp:
d  ==>     Inhaltsverzeichnis
-  ==>     "Normale" Datei
b  ==>     Blockorientiertes Gerät
c  ==>     Zeichenorientiertes Gerät
p  ==>     "Namend Pipe"
```

2.2.4 Allgemeine Kommandosyntax (nach GNU)

Viele GNU Kommandos unterstützen neben den einbuchstabigen Optionen die mit einem Minuszeichen beginnen, häufig auch sog. „Long Options“ die mit zwei Minuszeichen beginnen.

```
$  ls --help # Auszug!
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
  -b, --escape               print octal escapes for nongraphic characters
  -B, --ignore-backups       do not list implied entries ending with ~
  -C                         list entries by columns
      --color[=WHEN]         control whether color is used to distinguish file
                               types.  WHEN may be 'never', 'always', or 'auto'
  -d, --directory            list directory entries instead of contents,
                               and do not dereference symbolic links
  -F, --classify             append indicator (one of */=>@|) to entries
      --full-time            like -l --time-style=full-iso
      --group-directories-first
                             group directories before files
  -h, --human-readable       with -l, print sizes in human readable format
                               (e.g., 1K 234M 2G)
  -i, --inode                print the index number of each file
  -l                         use a long listing format
  -L, --dereference          when showing file information for a symbolic
                               link, show information for the file the link
                               references rather than for the link itself
  -m                         fill width with a comma separated list of entries
  -n, --numeric-uid-gid      like -l, but list numeric user and group IDs
  -q, --hide-control-chars   print ? instead of non graphic characters
      --show-control-chars   show non graphic characters as-is (default
                             unless program is 'ls' and output is a terminal)
      ....
```

2.3 Das Dateisystem

- Im Dateisystem werden alle Dateien eines Unix–Systems abgelegt
  Es werden drei verschiedene Dateiarten unterschieden:

Ordinary files	Normale Dateien (Programme, Textdateien, Datendateien) Sie können von beliebigen Kommandos bearbeitet werden (z.B. Editor).
Directory	Inhaltsverzeichnisdateien Sie werden von bestimmten Kommandos angelegt bzw. gelöscht und können von vielen Kommandos gelesen aber nicht beschrieben werden.
Special files	Gerätedateien Sind keine real existierenden Dateien, sondern stellen einen Gerätetreiber dar. Sie werden mit einem speziellen Kommando angelegt, können aber von beliebigen Kommandos gelesen bzw. beschrieben werden.

  Merke: Einem Kommando, dem als Parameter ein Dateiname übergeben wird, kann auch ein Gerätename (manchmal auch ein Inhaltsverzeichnis) übergeben werden. Ob das sinnvoll ist, hängt von dem Kommando und der angegebenen Datei ab.

- Ein Dateisystem besteht mindestens aus einem Verzeichnis, dem sogenannten Wurzelverzeichnis (root directory). Dieses hat den Namen "/".
- Jedes Verzeichnis kann beliebige Dateien enthalten (auch Verzeichnisdateien).
  Es beinhaltet mindestens zwei Einträge:
  - Der Name "." stellt einen Verweis auf das Verzeichnis selbst dar (current or working directory).
  - Der Name ".." stellt einen Verweis auf das übergeordnete Verzeichnis dar (parent directory).

  Daraus ergibt sich eine hierarchische Struktur.
- Durch eine Auflistung aller Verzeichnisse, ausgehend vom Wurzelverzeichnis, lässt sich jeder Punkt in dem hierarchischen System eindeutig beschreiben. Als Trennzeichen bei der Auflistung wird ein Slash ("/") verwendet. Eine solche Auflistung nennt man *absolute Pfadangabe*.
- Eine Auflistung, die nicht mit dem Wurzelverzeichnis beginnt, nennt man *relative Pfadangabe*. Sie startet implizit mit dem aktuellen Verzeichnis.
- Für jeden Benutzer wird vom Systemverwalter ein Verzeichnis eingerichtet, in das er beim Einloggen automatisch gelangt. Dieses Verzeichnis hat häufig den Namen /home/*loginname* und man nennt es Heimatverzeichnis (Home directory).

2.3.1 Begriffe zum Dateisystem

Aktuelles Verzeichnis (Current directory)
: Das Verzeichnis, in dem man sich gerade befindet (Name: "."). Alle relativen Pfadangaben beziehen sich auf dieses Verzeichnis. Der absolute Pfadname des aktuellen Verzeichnisses wird durch das Kommando `pwd` ausgegeben.

Elternverzeichnis (Parent directory)
: Das Verzeichnis, dass dem betrachteten Verzeichnis übergeordnet ist (Name: "..").

Wurzelverzeichnis (Root directory)
: Das Verzeichnis, dessen Elternverzeichnis auf sich selbst verweist (Name: "/").

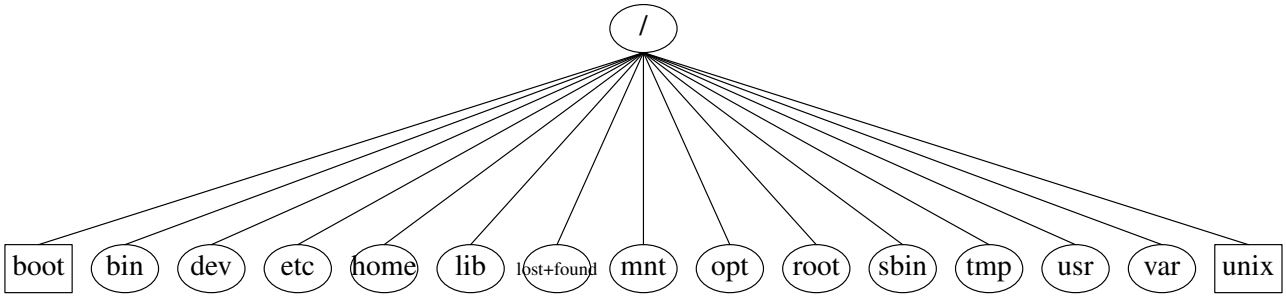
Loginverzeichnis
: Das Verzeichnis, in dem man sich direkt nach dem Einloggen befindet.

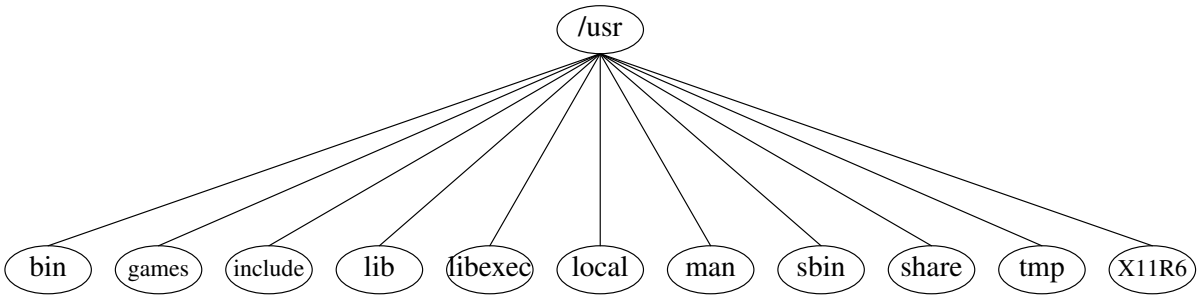
Heimatverzeichnis (Home directory)
: In dieses Verzeichnis wechselt man durch `cd` ohne Parameter. (Steht in der Variablen `$HOME`. Ist meistens identisch mit dem Loginverzeichnis.)

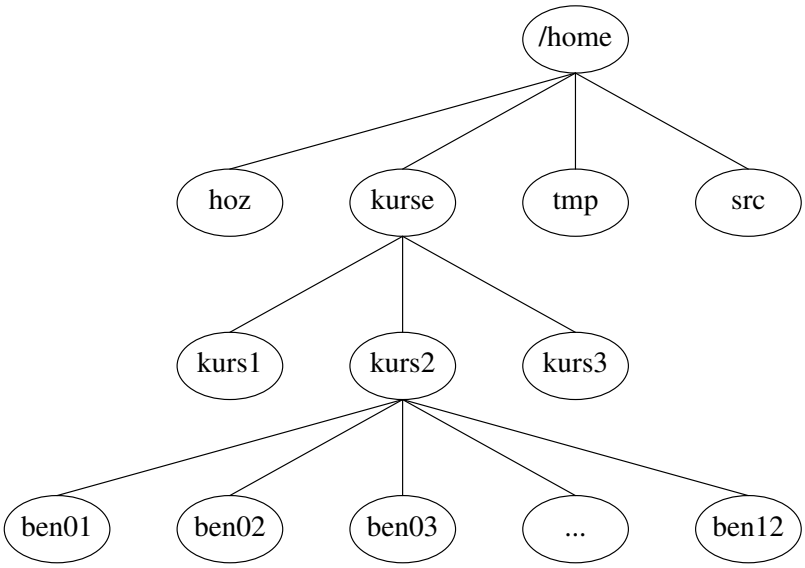
Absolute Pfadangabe
: Beginnt immer mit dem Namen des Wurzelverzeichnisse `("/")`.

Relative Pfadangabe
: Beginnt nie mit dem Namen des Wurzelverzeichnisses.

2.3.2 Beispiel für ein Dateisystem unter UNIX:







2.3.3 Kommandos zum Dateisystem

Kopieren von Dateien

cp (copy files)

Syntax:

cp [**–p**] *quelldatei zieldatei*

oder

cp [**–p**] [**–r**] *quelle1* [*quelle2 ...*] *zielverzeichnis*

- Kopiert die angegebene Datei und legt sie unter dem Zielnamen im Dateibaum ab.
- Kopiert alle angegebenen Quelldateien in das Zielverzeichnis.
- Wird die Option `-r` angegeben, werden alle Dateien und Verzeichnisse unterhalb der Quellangabe (rekursiv) kopiert. In diesem Fall kann *quelle* auch ein Verzeichnis sein.
- Durch die Option `-p` werden alle Zugriffs- und Besitzrechte der Quelldateien sowie der Zeitstempel übernommen. Um die Besitzrechte zu übernehmen, muss das Kommando als Systemverwalter ausgeführt werden.

Verweise auf Dateien anlegen

ln (link)

Syntax:

ln [**–s**] *alter_name zusätzlicher_name*

oder

ln [**–s**] *name1* [*name2 ...*] *zielverzeichnis*

- Erstellt einen neuen Namenseintrag für eine Datei. Die Datei ist danach unter einem weiteren Namen ansprechbar. Die Namen können auch in unterschiedlichen Verzeichnissen stehen.
- Wird die Option `-s` angegeben, wird ein symbolischer Link hergestellt. Dieser kann auch über Dateisystemgrenzen hinweg gehen.

Löschen von Dateien

rm (remove)

Syntax:

rm [**–irf**] *datei* [...] (remove)

- Löscht alle als Parameter angegebenen Dateien.
- Bei der Option `-i` wird vor jedem Löschen nachgefragt. (Antwort: y oder n)
- Bei der Option `-r` (rekursiv) wird ab dem angegebenen Pfad der gesamte Teilbaum einschließlich aller Unterverzeichnisse gelöscht.
- Durch die Option `-f` wird die Datei auch dann gelöscht, wenn kein Schreibrecht auf die Datei besteht, aber Schreibrecht auf das Verzeichnis existiert. Ohne die Option würde `rm` vor dem Löschen nachfragen.

Umbenennen (Bewegen) von Dateien

mv (move or rename files)

Syntax:

mv *alter_name neuer_name*

oder

mv *quelle1* [*quelle2 ...*] *zielverzeichnis*

- Gibt der angegebenen Datei einen neuen Namen, bzw. verschiebt sie in ein anderes Verzeichnis.
- Bewegt alle angegebenen Quelldateien in das Zielverzeichnis.

Ändern der Zugriffsrechte einer Datei

chmod (change modus)

Syntax:

chmod *modus[,modus] datei* [...]

- *modus* gibt die neuen Zugriffsrechte der Datei(en) an. Die Angabe kann in Form einer Oktalzahl oder in einer symbolischen Darstellung erfolgen.
- Bei der Angabe als Oktalzahl bestimmt die erste Ziffer der Oktalzahl die Zugriffsrechte des Besitzers, die zweite die der Gruppe und die letzte Ziffer die Zugriffsrechte für alle anderen Benutzer. Da eine Oktalziffer 8 verschiedene Werte annehmen kann, sind alle Kombinationen von Zugriffsrechten darstellbar.

user			*group*			*other*			Benutzerklasse
r	w	x	r	w	–	r	–	–	Symbolisch (ls -l)
1	1	1	1	1	0	1	0	0	Binär (Bitweise)
4	2	1	4	2	1	4	2	1	Gewichtung der Bits
7			6			4			Oktal

- Bei der symbolischen Angabe des Modus wird die Form "[Wer]Operator[Zugriffsrecht]" verwendet. Es können mehrere Modusangaben durch Komma getrennt hintereinander angegeben werden.

Wer	*Operator*	*Zugriffsrecht*
`u (user)`	`- (Recht wegnehmen)`	`r (read)`
`g (group)`	`+ (Recht geben)`	`w (write)`
`o (other)`	`= (Recht setzen)`	`x (execute)`
`a (all) (default)`		`s (set uid)`
		`t (save text)`

Die Bedeutung der Zugriffsrechte bei

Verzeichnissen:

Das Leserecht sorgt dafür, dass mit dem Kommando `ls` ein Verzeichnis aufgelistet werden kann.

Das Schreibrecht erlaubt das Anlegen und das Löschen(!) von Dateien in dem Verzeichnis, unabhängig von dem Schreibrecht auf die Datei.

Das Ausführungsrecht entscheidet darüber, ob auf die Dateien des Verzeichnisses zugegriffen werden kann.

Das t–Bit bewirkt, dass die Dateien in dem Verzeichnis nur von den Dateibesitzern gelöscht werden können.

Dateien:

Das Schreib- und Leserecht ist offensichtlich.

Das Ausführungsrecht muss bei den Dateien gesetzt werden, die als Kommando aufgerufen werden.

Das t–Bit bei ausführbaren Programmen sorgt dafür, dass nach Beendigung des Programmes eine Kopie im Swap Bereich bleibt. Das t–Bit darf nur beim Benutzer gesetzt werden (`u+t`).

- Der Modus einer Datei darf nur vom Dateibesitzer oder dem Superuser verändert werden.

  Beispiel:

```
$  ls -l bsp
-r-xr-x--x  1 hoz   kaho    4712  Jun 9 09:18  bsp

$  chmod ug+w,o-x bsp

$  ls -l bsp
-rwxrwx---  1 hoz   kaho    4712  Jun 9 09:18  bsp
```

- Die GNU Version des Kommandos kennt noch ein paar nützliche Optionen
  - — Rekursives Ändern der Zugriffsrechte über die Option `-R`.
  - — Bestimmen der zu setzenden Zugriffsrechte über eine Referenzdatei `--reference=`*FILE*

Ändern des Gruppennamens einer Datei

chgrp (change group)

Syntax:

chgrp [**–R**] *neuer_name datei* [...]

Beispiel:

```
$  ls -l bsp
-rwxrwx---  1 hoz   kaho    4712  Jun 9 09:18  bsp

$  chgrp other bsp

$  ls -l bsp
-rwxrwx---  1 hoz   other   4712  Jun 9 09:18  bsp
```

- Der Gruppenname darf nur von dem Superuser bzw. dem Dateibesitzer geändert werden.
- Über die Option `-R` werden die Gruppenrechte auch rekursiv in allen Unterverzeichnissen geändert.

Ändern des Dateibesitzers

chown (change owner)

Syntax:

chown [**–R**] *neuer_name datei* [...]

Beispiel:

```
$  ls -l bsp
-rwxrwx---  1 hoz   other   4712  Jun 9 09:18  bsp

$  chown root bsp

$  ls -l bsp
-rwxrwx---  1 root  other   4712  Jun 9 09:18  bsp
```

- Der Besitzer einer Datei darf nur von dem aktuellen Besitzer oder dem Superuser verändert werden.
- Über die Option `-R` werden die Gruppenrechte auch rekursiv in allen Unterverzeichnissen geändert.
- Neuere Versionen des Kommandos erlauben das gleichzeitige Ändern von Benutzer und Gruppenname. Die Syntax dafür lautet:
  Syntax:

  chown *neuer_user:neue_gruppe datei* [...]

Verändern des Dateidatums

touch (change time and date of files)

Syntax:

touch [**–am**] [**–r** *refdatei*] [**–t** *datum*] *file* [...]

- `touch` setzt das Datum (und die Zeit) der angegebenen Datei(en) auf das aktuelle Datum (und Zeit) oder auf das als Parameter zu der Option `-t` angegebene Datum. Die Datumsangabe erfolgt in der Form `[YYYY]MMDDhhmm[.ss]` wobei `YYYY` die Jahresangabe, `MM` der numerische Monat, `DD` die Tagesangabe und `hhmm` die Zeit, optional mit Angabe von Sekunden, ist.
- Unter Unix hat eine Datei drei unterschiedliche Zeitstempel. Durch `touch` werden die Access- und die Modification Zeitstempel gesetzt. Der Zeitpunkt des letzten Zugriffs kann nicht modifiziert werden.
- Die Optionen `-a` respective `-m` setzen jeweils nur die Access oder die Modification Time.
- Durch die Option `-r` wird die angegebene Datei als Referenzdatei verwendet.
- Bei der GNU Version des Kommanods kann das Datum über die Option `-d` auch als komplexer String z.B. "1. Juli 2009 10:30" angegeben werden.
  Syntax:

  touch [**–am**] [**–r** *refdatei*] { [**–t** *datum*] | [**–d** *datum*] } *file* [...]

Default Zugriffsrechte von Dateien bestimmen

Beim Anlegen von Dateien oder Verzeichnissen verwendet UNIX einen Wert von 666 (rw-rw-rw-) für Dateien bzw. 777 (rwxrwxrwx) für Verzeichnisse.

Möchte man diese Voreinstellung verändern, kann man sich eine Maske definieren, die von der Standardeinstellung subtrahiert wird. Diese Maske wird durch das Kommando `umask` gesetzt.

Beispiel:

```
$ umask 002
```

Dateien werden ab sofort mit den Zugriffsrechten 664 (rw-rw-r--) bzw. 775 (rwxrwxr-x) bei Verzeichnissen angelegt.

Wird das Kommando ohne Parameter aufgerufen, gibt es die aktuell eingestellte Maske aus. Die GNU Version des Kommando kann über die Option `-S` auch symbolische

Werte ausgeben.

ACHTUNG: Die Einstellung des Kommandos `umask` gilt nur in der aktuellen Sitzung.

Anlegen eines neuen Verzeichnisses

mkdir (make directory)

Syntax:

mkdir [**–p**] [**–m** *mode*] *name*

- Die Option `-p` sorgt dafür, dass Unterverzeichnisse die noch nicht existieren automatisch angelegt werden.
- Über die Option `-m` können die Zugriffsrechte des Verzeichnisses gesetzt werden. Eine Angabe synonym zu dem Kommando *chmod*(1) ist möglich.

Löschen eines Verzeichnisses

rmdir (remove directory)

Syntax:

rmdir *name* [...]

Suchen von Dateien in Verzeichnisbäumen

find (find files)

Syntax:

find *verzeichnis suchkriterium* [*aktion*]

- Das Kommando durchsucht den Dateibaum ab *verzeichnis* nach Dateien, die dem angegebenen Suchkriterium genügen.

  Als Suchkriterium sind folgende Angaben möglich:

`-name` *"name"*	Alle Dateien mit dem angegebenen Namen (Eine Musterangabe ist möglich).
`-group` *name*	Die Datei gehört der Gruppe *name*.
`-user` *name*	Die Datei gehört dem Benutzer *name*.
`-perm` *oktnum*	Die Zugriffsrechte der Datei sollen mit *oktnum* übereinstimmen.
`-size` *n*	Die Größe der Datei soll *n* Blöcke sein (Ein Block entspricht 512 Byte). -atime *n* T{ Auf die Datei soll in den letzten *n* Tagen zugegriffen worden sein.
`-ctime` *n*	Die Datei soll innerhalb der letzten *n* Tagen angelegt worden sein.
`-mtime` *n*	Die Datei soll in den letzten *n* Tagen verändert worden sein.
`-newer` *file*	Die Datei soll jüngeren Datums sein als die angegebene Datei.
`-type` *t*	Der Typ der Datei soll *t* sein, wobei *t* ein **b** ist für eine Blockgerätedatei, **c** für eine Charactergerätedatei, **d** für ein Inhaltsverzeichnis, **f** für eine FIFO - Gerätedatei und **p** für eine "Named Pipe".
`-o`	ODER-Verknüpfung der Suchkriterien.

- Zahlenwerte werden als Dezimalzahlen angegeben. Ein negatives Vorzeichen bedeutet dabei kleiner *n*, ein postives Vorzeichen größer *n*, kein Vorzeichen genau *n*.

- Die Auflistung der Suchkriterien stellt immer eine logische UND–Verknüpfung dar. Eine ODER–Verknüpfung lässt sich mit der Option `-o` erreichen, eine Negation durch das Zeichen "!".
  Eine Gruppierung von Ausdrücken ist durch runde Klammern möglich (Bitte durch "\" maskieren).

Als Aktion sind (unter anderen) die folgenden Angaben möglich

`-print`	Der Pfadname der gefundenen Dateien wird auf der Standardausgabe ausgegeben.
`-print0`	Eine sicherere Variante von -print, die auch bei Dateinamen funktioniert die z.B. Leerzeichen enthalten Diese Variante ist bestens geeignet um die Ausgabe zu dem Kommando *xargs*(1) zu senden. Bei diesem muss dann die Option `-0` gesetzt sein.
`-fprint` *datei* `-fprint0` *datei*	Wie `-print` resp. `-print0`. Die Ausgabe wird in der angegebenen Datei abgelegt (GNU Erweiterung).
`-exec` *cmd*	Für jede gefundene Datei, wird das Kommando *cmd* ausgeführt. Enthält die Kommandoangabe die Zeichenfolge {} wird diese durch den gefundenen Dateinamen ersetzt. Das Ende des Kommandos muss durch ein maskiertes Semikolen gekennzeichnet werden (\;).
`-cpio` *device*	Sichert alle gefundenen Dateien auf dem angegebenen Datenträger.

Beispiele:

Sucht nach der Datei "vi" und gibt den Pfadnamen auf dem Bildschirm aus.

```
$ find / -name vi -print
```

Ermittelt die Namen aller Unterverzeichnisse des Heimtaverzeichnisses.

```
$ find $HOME -type d -print
```

Löscht unterhalb von */home* alle Dateien mit dem Namen "core" oder "a.out" bzw. einer Dateigröße über 0.5 MByte.

```
$ find /home \( -name core -o -name a.out -o -size +1000 \) -exec rm {} \;
```

Achtung: Enthalten die gefundenen Datei- resp. Pfadnamen Leerzeichen, wird das obige Kommando gefährlich, da u.U. Dateien gelöscht werden die man nicht beabsichtigte. Besser ist es folgenden Aufruf zu verwenden

```
$ find /home \( -name core -o -name a.out -o -size +500K \) -print0 |
> xargs -0 rm
```

Sucht ab */usr* nach Dateien mit gesetztem S-Bit und speichert deren Namen in der Datei *tmp/suid.files*. Gleichzeitig werden die Namen aller Dateien die Größer als 100 MB sind in der Datei *tmp/big.files* abgelegt.

```
$  find /usr   \( -perm -4000 -fprint ~/tmp/suid.files \) , \
               \( -size +100M -fprint ~/tmp/big.files \)
```

Sucht in allen C-Programmdateien nach dem Wort *variable*.

```
$ find $HOME \( -name '*.c' -o -name '*.h' \) -print0 | \
          xargs -0 grep -w variable
```

Wer obiges öfter machen möchte, dem sei die Installation von *rg*(1) nahegelegt (siehe Kapitel *Dateibearbeitung*).

```
$ rg -t c -w variable
```

2.4 Exkurs: Editorbenutzung

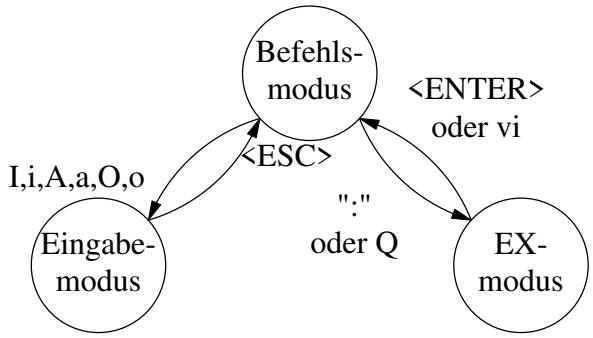
In UNIX gibt es im wesentlichen vier verschiedene Editoren.

- `ed` → Zeileneditor.
- `ex` → Zeileneditor mit größerem Befehlsumfang.
- `vi` → Bildschirmeditor, der auf "ex" aufgebaut ist.
- `emacs` → **e**ight **m**egabit **a**nd **c**onstantly **s**wapping :-)

Viele (Linux)Systeme bringen heutzutage auch einfachere, bildschirmorientierte Editoren (z.B. *pico*(1) oder *nano*(1)) mit.

Obwohl die Arbeitsweise des `vi` sehr ungewöhnlich ist, sollte man sich mit dem Editor bechäftigen und ihn bedienen lernen, da er sozusagen der Standardeditor unter Unix ist. Bei GNU Systemen ist in der Regel `vim` installiert, der noch leistungsfähiger, und dabei abwärtskompatibel zu *vi* ist.

Der `vi` besitzt drei verschiedene Arbeitsmodi:



2.4.1 Überblick über die wichtigsten Editor Kommandos

Die unentbehrlichen Kommandos sind mit * gekennzeichnet.

Aufruf des Editors

Syntax:

vi [**–rRc**] *dateiname* [...]

Beenden des Editors

	`:w`	(write)	Sichert die aktuelle Datei
	`:w [datei]`		Sichert die Datei unter dem angeg. Namen
	`:wq`	(write and quit)	Sichert die aktuelle Datei und verlässt "vi"
*	`ZZ`		siehe ":wq"
*	`:q!`	(quit)	Verlässt "vi" ohne abzuspeichern

Umschalten in den Eingabemodus

*	`i`	(insert)	Einfügen vor dem Cursor
	`I`		Einfügen am Zeilenanfang
	`a`	(append)	Einfügen nach dem Cursor
*	`A`		Einfügen am Zeilenende
	`o`	(open)	Einfügen einer Zeile nach der aktuellen Zeile
	`O`		Einfügen einer Zeile vor der aktuellen Zeile

Cursorpositionierung

	`l`		Cursor ein Zeichen nach rechts
*	`<SPACE>`		
	`h`		Cursor ein Zeichen nach links
*	`<BACKSP>`		
	`j`		Cursor eine Zeile nach unten
*	`<ENTER>`		
*	`k`		Cursor eine Zeile nach oben
*	`w`	(word)	Cursor auf das nächste Wort (Sonderzeichen zählen als Wort)
	`W`		Wie bei "w" (Wort ist ein durch *white space* getrenntes Wort)
*	`b`	(back)	Vorheriges Wort oder Sonderzeichen
	`B`		Vorheriges Wort
	`e`	(end)	An das Ende des Wortes oder Sonderzeichen
	`E`		An das Ende des Wortes
	`^`		Zum Anfang der Zeile
*	`$`		Zum Ende der Zeile
	`H`	(high)	Zum Anfang des Bildschirms
	`M`	(middle)	In die Mitte des Bildschirms
	`L`	(low)	Zum Ende des Bildschirms
*	*n*`G`	(go)	Zur Zeile *n*
	`G`		Zur letzten Zeile
	`^B`	(back)	Eine Seite zurück blättern
	`^F`	(forward)	Eine Seite vorwärts blättern
*	`^U`	(up)	Eine halbe Seite rückwärts rollen
*	`^D`	(down)	Eine halbe Seite vorwärts rollen
	`^D`		Eine Tabulatorposition nach links gehen (Im Eingabemodus)

Text löschen

*	`x`	(ausixen)	Löscht das Zeichen unter dem Cursor
	`X`		Löscht das Zeichen vor dem Cursor
*	`dw`	(delete word)	Löscht das nächste Wort
*	`dd`		Löscht die aktuelle Zeile
	`D`		Löscht bis zum Zeilenende

Text ändern

`r`*zeichen*	(replace)	Ersetzt das Zeichen unter dem Cursor durch das angegebene Zeichen
`s`*zeichenfolge*`<ESC>`	(substitute)	Ersetzt das Zeichen unter dem Cursor durch die Zeichenfolge
`cw`*zeichenfolge*`<ESC>`	(change word)	Ersetzt das Wort im Text durch das angegebene Wort (Bis zur Eingabe von <ESC>)

Suchen von Zeichenketten

*	`/Muster`	"Muster" vorwärts suchen

	`?Muster`	"Muster" rückwärts suchen
*	`n`	Nächstes Auftreten von *Muster* suchen
*	`N`	letztes vorhergehendes Auftreten von *Muster* suchen
*	`%`	Entsprechende Klammer '()', '[]', '{}' suchen Der Cursor muss dabei auf der Klammer stehen

Suchen und Ersetzen

:[von_Zeile,bis_Zeile]`s`/*suchmuster*/*ersetzungsmuster*/[`gc`]

- Die eckigen Klammern gehören nicht zur Syntax. Sie kennzeichnen lediglich optionale Angaben.
- "von_Zeile" und "bis_Zeile" sind Zeilennummernangaben. (Ohne sie wird das Kommando nur auf die aktuelle Zeile angewendet.)
- Das Flag `g` gibt an, dass jedes Auftreten von *Suchmuster* durch *Ersetzungsmuster* ersetzt werden soll. (Standard: Nur das erste Auftreten in der Zeile.)
- Das Flag `c` erlaubt interaktives Ersetzen.

Markieren von Textteilen

`Y`	(großes Y)	Kopiert die aktuelle Zeile in den internen Puffer
`yw`	(kleines Y)	Kopiert das nächste Wort in den internen Puffer

Einfügen von gelöschten oder markierten Textteilen

*	`p`	(kleines p)	Einfügen des zuletzt gelöschten bzw. markierten Textes hinter den Cursor
*	`P`	(großes P)	Einfügen des zuletzt gelöschten bzw. markierten Textes vor den Cursor

Anhängen von Zeilen

*	`J`	(join)	Hängt die nächste Zeile an die aktuelle Zeile an

Aufheben von Änderungen

*	`u`	(undo)	Macht die letzte Änderung (einfügen, löschen etc.) rückgängig
	`U`		Stellt die zuletzt geänderte Zeile wieder her

Anmerkung:

Die meisten Editorkommandos können durch die Eingabe einer Zahl vor dem Kommando mehrfach ausgeführt werden.

Beispiel:

`3dd`	Löscht die nächsten drei Zeilen.
`2w`	Bewegt Cursor um 2 Worte nach rechts.

Empfehlenswerte VI Standardeinstellungen

Der Editor `vi` lässt sich über Schalter in seiner Arbeitsweise beeinflussen. Die Schalter werden über das Kommando `set` gesetzt bzw. zurückgesetzt. Das Zurücksetzen eines Schalter erfolgt durch voranstellen der Silbe `no` vor dem Schalternamen.
Damit diese Schaltereinstellungen nicht nach jedem Start des Editors erneut durchgeführt werden müssen, können alle `set`–Kommandos in einer Datei mit dem Namen `.exrc` abgelegt werden. Der `vi` sucht nach einer solchen Datei im Login– und im

aktuellen Verzeichnis.

`set report=1`	Meldungen werden ausgegeben, wenn sich ein Kommando auf mehr als eine Einheit bezieht.
`set showmode`	Zeigt den jeweiligen Modus an in dem sich "vi" befindet.
`set number`	Zeigt Zeilennummern mit an.
`set autoindent`	Automatisches Einrücken (Sinnvoll bei der Programmentwicklung).
`set autowrite`	Automatisches Sichern beim Wechseln der bearbeiteten Datei.

Übungen zum Editor VI

1. Legen Sie eine Datei mit dem Namen `.exrc` in Ihrem Heimatverzeichnis an. Die Datei sollte einige oder alle `set`–Kommandos des vorigen Abschnittes enthalten.
2. Übertragen sie die Dateien `t1f.c`, `t2f.c` und `t3f.c` von dem Webserver `http://www.hznet.de` aus dem Verzeichnis `pub` in ihr Heimatverzeichnis. Sie sollen dort unter den Namen `t1.c`, `t2.c` und `t3.c` verfügbar sein. Suchen sie im Stichwortverzeichnis nach der Beschreibung des Kommandos *wget*(1) oder *curl*(1) und benutzen sie dieses um die Daten von dem Webserver zu übertragen.[5]
3. Führen Sie Veränderungen an den Dateien aus, so dass die im folgenden abgedruckten Dateien entstehen.

   t1.c
   Aus dem großen `M` in Zeile 5 ein kleines `m` machen. Am Ende der Zeile 7 ein Semikolon anfügen. Nach der Zeile 7 eine Zeile mit einer geschweiften Klammer zu einfügen.

```
1    /* Mein erstes C-Programm */
2
3    # include <stdio.h>
4
5    main()
6    {
7    printf("Hello world \n");
8    }
```

   t2.c
   Am Anfang der Zeile 5 ein `c` einfügen. In Zeile 7 aus dem `z` ein `e` machen. In Zeile 8 die Zeichen `st` vertauschen. Die Zeilen 5–8 um eine Tabulatorposition einrücken.

```
1    # include <stdio.h>
2
3    main()
4    {
5         char s[80+1];
6
7         if ( gets(s) )
8              puts(s);
9    }
```

   t3.c

5. Die anzugebene URL sieht dann z.B. so aus: http://www.hznet.de/pub/t1f.c

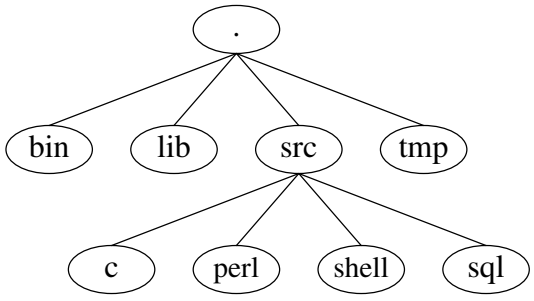
In Zeile 3 `t` und `n` vertauschen. Zeile 4 löschen. In Zeile 5 das `u` von `au` löschen. Die Zeilen 5 und 6 vertauschen.

```
1     main()
2     {
3     int  a,b;
4
5     scanf("%d%d",&a,&b);
6     printf("%f\n",b/(double)a);
7     }
```

4. Kompilieren Sie jede der Dateien mit den folgenden Kommandos:
```
$  cc t1.c -o hello
$  cc t2.c -o firstline
$  cc t3.c -o quotient
```
5. Starten Sie die Kommandos
```
$  hello
$  firstline
$  quotient
```

2.5 Aufgaben

1. Erstellen Sie in Ihrem Heimatverzeichnis folgenden Verzeichnisbaum:



2. Suchen Sie auf der Platte die Datei `books` und linken Sie sie in Ihr `lib` Verzeichnis (Hinweis: Symbolische Links).
3. Suchen Sie ab dem Verzeichnis `/home/kurse` nach der Datei `banner` und kopieren Sie sie in Ihr `bin` Verzeichnis. Setzen sie die Ausführungsrechte für die Datei.
4. Legen Sie in Ihrem `bin` Verzeichnis eine Datei `cls` an, die den Text `tput clear` enthält. Setzen Sie die Ausführungsrechte der Datei.
5. Kopieren Sie die Datei `/etc/passwd` unter dem Namen `ulist` in Ihr Verzeichnis `lib`. Löschen Sie in jeder Zeile alle Zeichen ab dem 5. Doppelpunkt (einschließlich). Löschen Sie in jeder Zeile alle Zeichen ab dem 1. Doppelpunkt (ausschließlich) bis zum 4. Doppelpunkt (einschließlich).
6. Bewegen Sie die Kommandos `hello`, `firstline` und `quotient` in Ihr `bin` Verzeichnis, und die zugehörigen C–Quelltextdateien in Ihr Verzeichnis `src/c`.

3. Der Kommandointerpreter

Der Kommandointerpreter ist der Teil eines UNIX–Systems, der die Schnittstelle des Betriebsystemkerns zum Benutzer darstellt. Mit seiner Hilfe ist es dem Benutzer möglich Kommandos auszuführen. Der Kommandointerpreter legt sich also wie eine Schale um den Betriebsystemkern. Aus diesem Grunde wird er häufig auch *Shell* genannt.

Die Shell ist ein Kommando, das beim Einloggen eines Benutzers automatisch gestartet wird, ein Promptsymbol auf den Bildschirm schreibt und auf die Eingabe eines Kommandos wartet.

In neueren UNIX–Systemem stehen — zusätzlich zum Standardkommandointerpreter (Bourne Shell) — weitere Kommandointerpreter zur Verfügung.

Der Kommandointerpreter, der auf allen UNIX–Systemen zur Verfügung steht, wurde von Steven Bourne entwickelt und steht als ausführbares Kommando im Verzeichnis `/bin` unter dem Namen `sh`.

Die Bourne Shell war die erste verfügbare Shell, wurde 1977/1978 entwickelt und ist dementsprechend nicht besonders komfortabel.
Die größten Kritikpunkte:

— Keine komfortable Editiermöglichkeit in der Kommandozeile

— Keine Möglichkeit der Wiederholung von Kommandos (history Mechanismus)

— Keine Pfadangabe im Prompt möglich.

Inzwischen ist auf fast allen UNIX–Plattformen ein Kommandointerpreter verfügbar, auf den die oben genannten Kritikpunkte nicht zutreffen, der aber kompatibel zur Bourne Shell ist. Dieser Kommandointerpreter wurde von David G. Korn entwickelt und hat daher den Namen Korn Shell (`/bin/ksh`). Alle GNU basierten Unix Systeme nutzen jedoch eine ksh kompatible Shell, die darüber hinaus noch mehr Möglichkeiten bietet. Diese Shell nennt sich Bourne again Shell (`bash`).

3.1 Automatischer Start der Shell

Die Shell wird automatisch beim Einloggen gestartet. Welche Shell dies ist wird, durch den Systemverwalter für jeden Nutzer separat eingestellt. Sowohl Korn Shell als auch Bash verhalten sich nach außen wie eine Bourne Shell. Erst durch das Setzen einiger Optionen bzw. Shellvariablen werden die neuen Möglichkeiten sichtbar.

Zur einfacheren Handhabung wird bei jedem Unix System eine Datei mit den wichtigsten Einstellungen zur Shell mitgeliefert und im Benutzerverzeichniss installiert.

Für die Korn Shell heißt diese Datei `.kshrc` und für die Bash `.bashrc`.

In der Datei `.profile` im Loginverzeichnis kann eine Zeile mit dem Text

```
set -o ignoreeof
```

eingefügt werden, damit die Loginshell nicht durch die Taste `<^D>` beendet werden kann.

3.2 Bedienung der Korn Shell / Bash

Die Korn Shell als auch die Bash erlauben das Editieren der Komandozeile bevor diese abgesendet wird. Auch eine Historie bereits abgesetzter Kommandos wird verwaltet, sodass man auf alte Kommandos zurückgreifen, diese editieren, und erneut ausführen kann.

Für das Editieren werden zwei unterschiedliche Modi vewendet:

1. Im EMACS Modus können die Cursortasten verwendet werden. In der Regel ist dies der voreingestellte Modus.
2. Im VI Modus sind die gleichen Kommandos wie im Editor *vi*(1) nutzbar. Wer diesen Editor gut bedienen kann oder mit ihm arbeitet, sollte die Komanndozeileneditierung auf diesen Modus umschalten.

Die Option `-o vi` bei Aufruf der Korn Shell sorgt dafür, dass die Tasten, die zur Editierung der Kommandozeile verwendet werden, mit den Tasten des Editors `vi` übereinstimmen.

- Standardmäßig befindet man sich im Appendmodus, d.h. jedes getippte Zeichen wird am Ende der Zeile angehängt. Die Taste `<BackSpace>` zum Löschen des zuletzt eingegebenen Zeichens ist wirksam.
- Mit `<ESC>` wechselt man in den Befehlsmodus. Anschließend sind die Cursorpositionierungskommandos des `vi` wirksam.
- Durch ein Einfügekommando (`I,i,A,a,r,s,c`) gelangt man erneut in den Eingabemodus.
- Die Taste `<ENTER>` führt immer das in der Zeile angezeigte Kommando aus. `<DEL>` verwirft die Zeile und stellt eine leere Eingabezeile zur Verfügung.
- Das eingebaute Kommando `history` zeigt die letzten 16 Kommandos des History Puffers an.
- Mit dem Kommando *k* bewegt man sich innerhalb des Historypuffers nach oben, mit *j* nach unten.

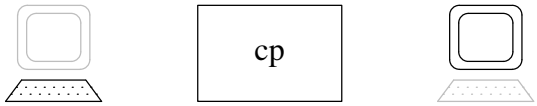
4. Befehlsformate

4.1 Klassifizierung von Kommandos

4.1.1 Kommandos ohne Ein/Ausgaben

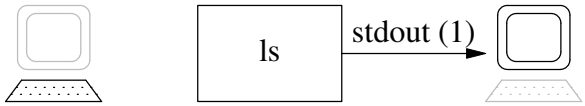
Einige der bisher vorgestellten Kommandos erzeugen keinerlei Ausgabe auf dem Bildschirm (außer vielleicht einer Fehlermeldung).

Sie benötigen für ihre Aufgabe auch keine weiteren Eingaben, außer den Parametern die man beim Aufruf des Kommandos angibt. Solche Kommandos (cp, mkdir, mv) kann man graphisch wie folgt darstellen:



4.1.2 Kommandos mit Ausgaben (Produzenten)

Zwei der bisher vorgestellten Kommandos (ls, find) erzeugen eine Ausgabe, die im Normalfall auf den Bildschirm ausgegeben wird. Kommandos dieser Art nennt man *Produzenten*, da sie Ausgaben auf der sogenannten *Standardausgabe* (stdout) erzeugen. Die Standardausgabe ist im Normalfall mit dem Bildschirm verbunden.

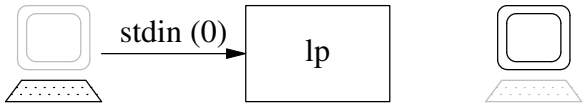


Weitere Beispiele für Produzenten sind:

```
$  date
Sun Aug 16 08:29:59 CEST 2009

$  who
hoz      pts/0        Aug 12 16:37
hoz      pts/3        Aug 12 16:37
hoz      pts/4        Aug 16 08:11
gustav   pts/1        Aug 06 16:58

$  banner Hallo
#     #
#     #    ##    #       #        ####
#     #   #  #   #       #       #    #
#######  #    #  #       #       #    #
#     #  ######  #       #       #    #
#     #  #    #  #       #       #    #
#     #  #    #  ######  ######   ####

$  echo Hallo Leute
Hallo Leute

$  logname
hoz

$ seq -s " " 10 5 100
10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
```

4.1.3 Kommandos mit Eingaben

Eine dritte Art von Kommando benötigt für ihre Tätigkeit Eingaben, die im einfachsten Fall über die Tastatur einzugeben sind. Diese Kommandos geben allerdings nichts auf dem Bildschirm aus.



Ein (seltenes) Beispiel dafür ist das Programm (`lp`), zur Druckausgabe. Die zu druckenden Daten werden über die Tastatur eingegeben. Die Eingabe wird mit der Taste `<^D>` am Zeilenanfang beendet.

```
$  lp
Dies ist der Text,
der auf dem Drucker
ausgegeben wird.
^D
$
```

Man sagt: das Kommando liest von der *Standardeingabe*. Solche Kommandos nennt man *Konsumenten*.

4.1.4 Kommandos mit Ein– und Ausgaben

Die vierte Art von Kommando liest von ihrer Standardeingabe (Tastatur) Daten ein, verarbeitet diese und gibt sie anschließend (in veränderter Form) auf der Standardausgabe (Bildschirm) wieder aus.

Solche Kommandos werden *Filter* genannt, da sie ihre Eingabedaten modifiziert wieder ausgeben. Ein typisches Beispiel ist das Kommando `sort`, das seine Eingabe zeilenweise sortiert wieder ausgibt.

Die meisten UNIX–Programme sind Filterprogramme.

Ein weiteres Beispiel für ein Filterprogramm ist `wc`. Es zählt die Zeichen, Zeilen und Worte seiner Eingabe und gibt das Ergebnis auf der Standardausgabe aus.

4.1.5 Aufgabe

Ordnen sie die folgenden Kommandos nach dem oben dargestellten Schema ein:

```
hello
firstline
quotient
cat
```

4.2 Umlenkung der Ein- und Ausgabe

4.2.1 Eingabeumlenkung

Bei Kommandos die von der Standardeingabe lesen, kann angegeben werden, dass die Eingabedaten nicht von der Tastatur sondern aus einer Datei gelesen werden.

Von dieser Eingabeumlenkung merkt das Kommando nichts. Es „denkt“ nach wie vor, dass die Daten über die Tastatur eingelesen werden.

Die Eingabeumlenkung erfolgt durch ein Kleinerzeichen gefolgt von einem Dateinamen.

```
$  kommando < dateiname
```

Beispiele:

```
$  sort < lib/ulist
$  lp < src/c/t2f.c
$  wc < src/c/t2f.c
$  firstline < lib/ulist
```

4.2.2 Ausgabeumlenkung

Analog zur Eingabeumlenkung kann bei Kommandos die Ausgaben auf der Standardausgabe produzieren (Produzenten, Filter), die Ausgabe statt auf den Bildschirm in eine Datei geschrieben werden.
Auch diese Umlenkung beeinflußt die Arbeitsweise des Kommandos selbst nicht.

Die Umlenkung der Ausgabe eines Kommandos erfolgt über ein Größerzeichen gefolgt von einem Dateinamen.

```
$  kommando > dateiname
```

Beispiele:

```
$  ls -R > files
$  who > wlist
$  hello > meld
```

Die Datei wird durch die Ausgabeumlenkung automatisch angelegt. Existiert die Datei bereits wird ihr Inhalt überschrieben.

Anhängen an Dateien

Soll die Ausgabe eines Kommandos an eine bestehende Datei angehängt werden, kann dies durch zwei Größerzeichen gefolgt von einem Dateinamen erreicht werden.

```
$  kommando >> dateiname
```

Beispiele:

```
$  pwd > files     # Name des akt. Verz. in die Datei schreiben
$  ls >> files     # Liste der Dateien anhängen
```

Existiert die Ausgabedatei nicht, wird sie automatisch angelegt.

4.2.4 Gleichzeitige Umlenkung der Ein/Ausgabe

Bei Filterprogrammen kann sowohl die Eingabe als auch die Ausgabe mit einer Datei verbunden werden.

```
$  kommando  < Eingabedatei  > Ausgabedatei
$  kommando  > Ausgabedatei  < Eingabedatei
```

Beispiel:

```
$  sort < wlist > swlist
```

Die Reihenfolge der Umlenkungszeichen ist beliebig.

4.2.5 Verketten von Kommandos (Pipelining)

Häufig soll die Ausgabe eines Kommandos von einem anderen Kommando weiterverarbeitet werden.

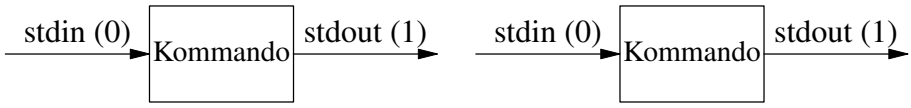
Beispiel:

```
$  who > temp
$  sort < temp
$  rm temp
```

Die Ausgabe des Kommandos `who` (Eine Liste aller zur Zeit eingeloggten Benutzer) wird in einer Zwischendatei abgelegt. Diese Datei wird durch das Kommando `sort`

zeilenweise sortiert und das Ergebnis auf die Standardausgabe geschrieben. Anschließend muss die Zwischendatei gelöscht werden.

Dieses Verfahren ist nicht nur sehr umständlich sondern auch zeitintensiv, da das Kommando `sort` erst mit der Arbeit beginnen kann, wenn das Kommando `who` abgeschlossen ist. Im Multitasking–Betrieb liegt die Idee nahe beide Kommandos parallel ablaufen zu lassen und eine Kopplung der Standardausgabe des einen Kommandos mit der Standardeingabe des zweiten Kommandos zu realisieren.



Ein solche Kopplung erreicht man über eine Pipeline (Rohrleitung). Als Symbol wird dazu der senkrechte Strich verwendet (Ein veraltetes Symbol ist das Caret).

```
$  Kommando1 | Kommando2
```

Beispiel:

```
$  who | sort
```

Die Verkettung von Kommandos über Pipelines kann beliebig erweitert werden.

Beispiel:

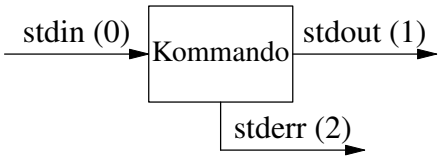
```
$  who | sort | lp
```

Regeln zu Pipelines:

- Jede Komponente einer Pipeline muss ein Kommando sein (keine Datei).
- Am Anfang einer Pipeline (links von einem Pipezeichen) dürfen nur Kommandos stehen die auf die Standardausgabe schreiben (Produzenten oder Filter).
- Am Ende einer Pipeline (rechts von einem Pipezeichen) dürfen nur Kommandos stehen die von der Standardeingabe lesen (Konsumenten oder Filter).
- In der Mitte einer Pipeline (zwischen zwei Pipezeichen) dürfen nur Kommandos stehen die sowohl von der Standardeingabe lesen als auch auf die Standardausgabe schreiben (Filterprogramme).
- Steht am Anfang einer Pipeline ein Filterprogramm kann die Eingabe aus einer Datei umgelenkt werden.
  ```
  $  sort < daten | lp
  ```
- Steht am Ende einer Pipeline ein Filterprogramm kann die Ausgabe in eine Datei umgelenkt werden.
  ```
  $  who | sort > daten
  ```

4.2.6 Umlenkung der Fehlerausgabe

Zusätzlich zu der Standardausgabe existiert ein weiterer Ausgabekanal der im Normalfall mit dem Bildschirm verbunden ist.



Auf diesem Kanal werden alle Fehlermeldungen von Kommandos ausgegeben. Dadurch werden Fehlermeldungen auch dann noch auf dem Bildschirm dargestellt, wenn die Standardausgabe in eine Datei umgelenkt wurde. Der Kanal wird daher

Standardfehlerausgabe (stderr) genannt.

Beispiel:

```
$ ls -l /non/existent/dir > files
ls: cannot access '/non/existent/dir': No such file or directory
```

Bei der Angabe des Umlenkungszeichens kann die Kanalnummer mit angegeben werden. Durch dieses Konzept können Fehlermeldungen und normale Ausgaben getrennt umgelenkt werden.

```
$  ls -l /non/existent/dir > files 2> error
$  cat < error
ls: cannot access '/non/existent/dir': No such file or directory
```

Beispiele:

```
# Fehlermeldungen des C-Compilers in einer Datei ablegen
$  cc bsp.c 2> error
# Fehlermeldungen wegwerfen
$  find /home -name t1f.c -print 2> /dev/null
```

4.2.7 Mischen von Standardausgabe– und Standardfehlerkanal

Es kann sinnvoll sein die Trennung der beiden Kanäle aufzuheben um beide Kanäle gemeinsam in eine Datei umzuleiten.

Dies geschieht über die Zeichenfolge `2>&1`.

Beispiel:

```
$  cc bsp.c 2>&1 | lp    # Fehlermeldungen des Compilers auf Drucker ausgeben
$  find /home -name t1f.c -print > ergebnis 2>&1
```

4.3 Mehrere Kommandos in einer Eingabezeile

Es können in einer Zeile mehrere, durch Semikolon getrennte Befehle eingegeben werden.

Diese werden dann der Reihe nach abgearbeitet.

Beispiel:

```
$  date; who
$  sleep 20; echo 20 Sekunden sind um
```

Soll die Ausgabe einer solchen Kommandosequenz gemeinsam in eine Datei gelenkt werden, muss die Sequenz geklammert werden.

Beispiel:

```
$  { date; who; } > who_now
oder
$  {
>  date
>  who
>  } > who_now
```

ACHTUNG:

— Nach "{" muss ein Leerzeichen oder ein Zeilentrenner stehen.

— Vor "}" muss ein Semikolon oder ein Zeilentrenner stehen.

4.4 Befehlssubstitution

Mit Hilfe der Befehlssubstitution ist es möglich, die Ausgabe eines Kommandos als Parameter eines anderen Kommandos zu verwenden.

Eine Kommandofolge, die in umgekehrte Hochkomma eingeschlossen ist, wird ausgeführt und das Ausführungsergebnis an dieser Stelle in die Kommandozeile eingefügt.

Beispiele:

```
$  echo Sie befinden sich im Verzeichnis `pwd`
Sie befinden sich im Verzeichnis /home/hoz

$  banner `logname`

$  banner `date +"%T"`

$  vi `find . -name "t1.c" -print`
```

4.5 Variablen

Der Kommandointerpreter ermöglicht die Verwendung von Variablen innerhalb einer Kommandozeile.

4.5.1 Zugriff auf Variablen

Um auf einen Variablenwert zuzugreifen schreibt man ein Dollarzeichen vor den Variablennamen (Das Dollarzeichen ist der Zugriffsoperator). Der Kommandointerpreter ersetzt dies durch den Inhalt der Variable.

Variablennamen können aus Klein- bzw. Großbuchstaben, Ziffernzeichen oder dem Unterstrich bestehen.
Kann der Variablenname nicht einedutig abgegrenzt werden, erfolgt der Zugriff auf die Variable durch die Form `${Variablenname}`.

Einige Variablen sind bei jedem Unix System mit Werten vorbelegt. Die Namen dieser Standardvariablen bestehen in der Regel nur aus Großbuchstaben.

Beispiele:

```
$ echo $LOGNAME
hoz
     # Ausgabe der Variablen LOGNAME_user (leer)
$ echo $LOGNAME_user
     # Ausgabe der Variablen LOGNAME gefolgt von _user
$ echo ${LOGNAME}_user
hoz_user

$ cp a.c b.c $HOME/src/c
```

4.5.2 Definition von Variablen

Selbstdefinierte Variablen werden angelegt indem ihnen ein Wert zugewiesen wird.

Die Zuweisung erfolgt mit Hilfe des Gleichheitszeichens, dass direkt hinter den Variablennamen (ohne Dollarzeichen) geschrieben wird.
Der zuzuweisende Wert muss in Anführungszeichen eingeschlossen werden falls er Leerzeichen enthält. Er muss direkt hinter dem Gleichheitszeichen stehen.

Beispiel:

```
$  a="Dies ist ein Beispiel"
$  v=`pwd`
```

4.6 Platzhaltersymbole (Wildcard)

4.6.1 Datei– und Verzeichnisnamen unter UNIX

- Bestehen aus maximal 14 Zeichen (SysV). Bei neueren Systemen (ab SysV.4) bzw. BSD–Unix können Dateinamen bis zu 255 Zeichen lang sein.
- Dateinamen können beliebige Zeichen enthalten (auch nicht sichtbare Zeichen z.B. `<ESC>`). Diese können beim Zugriff auf die Datei zu Problemen führen und sollten daher vermieden werden. Dies gilt insbesondere auch für Leerzeichen in Datei- und Pfadnamen.
- Groß- und Kleinbuchstaben werden unterschieden.
- Der Punkt ist ein „normales“ Zeichen (MS_DOS: Trennzeichen zwischen Name und Extension) und kann beliebig oft in einem Dateinamen auftreten.
- Dateinamen die mit einem Punkt beginnen werden bei `ls` nicht mit angezeigt. Auf sie werden auch nicht die Platzhaltersymbole angewendet.
- Pfadangaben werden durch Slash getrennt.
- Bei der Angabe von Dateinamen können Zeichen verwendet werden die eine Platzhalterfunktion besitzen.

4.6.2 Platzhaltersymbole

Als Platzhaltersymbole[6] sind folgende Zeichen erlaubt:

`*`	Steht für eine beliebige (auch leere) Folge von Zeichen
`?`	Steht für genau ein beliebiges Zeichen
`[..]`	Steht für eines der in Klammern angegebenen Zeichen
`[!..]`	Steht für keines der in Klammern angegeben Zeichen
`[..-..]`	Minuszeichen zwischen zwei Zeichen in eckigen Klammern: Ersetzt den Bereich zwischen den beiden Zeichen (von..bis – Angabe z.B. `[a-z]`)

Alle Platzhaltersymbole wirken nicht auf Dateien deren Namen mit einem Punkt beginnen.

Alle Platzhaltersysmbole können mehrfach in einem Muster vorkommen.

Der Kommandointerpreter ersetzt das Muster durch alle Namen des aktuellen Verzeichnisses auf die das Muster passt bevor das Kommando ausgeführt wird. Aus diesem Grunde arbeiten alle Unix Kommandos die mehrere Dateien verarbeiten können auch automatisch mit Platzhaltersysmbolen.

Beispiele:

`*`	Alle Dateinamen des aktuellen Verzeichnisses (ohne die Dateien, die mit einem Punkt beginnen)
`a*b`	Alle Dateinamen, die mit "a" beginnen und mit "b" enden
`a*b*c`	Alle Dateinamen, die mit "a" beginnen, mit "c" enden und irgendwo dazwischen ein "b" enthalten
`a?b`	Alle Dateinamen, die mit "a" beginnen, mit "b" enden und genau drei Zeichen groß sind
`*[0-9]` oder `*[0123456789]`	Alle Dateinamen, die mit einer Ziffer enden

6. Auch als Muster, Joker oder mit dem Englsichen Wort Wildcard bezeichnet

`[A-Z]*` Alle Dateinamen, die mit einem Großbuchstaben beginnen

`*.c` Alle Dateinamen, die mit ".c" enden

`.[!.]*` Alle Dateinamen, die mit "." beginnen und danach mindestens ein Zeichen besitzen, dass jedoch kein Punkt ist

Achtung:
Bei der Ein/Ausgabeumlenkung werden die Platzhaltersymbole nicht durch die entsprechenden Dateinamen ersetzt. Durch das Kommando

```
$  cat > *.bsp
```

wird eine Datei mit dem ungebräuchlichen Namen `*.bsp` angelegt.

Aufgabe:

Wie kann die Datei mit dem problematischen Namen `*.bsp` gelöscht werden (Bitte nur diese!)?

4.7 Maskierungssymbole

Da es oftmals notwendig ist Sonderzeichen als normale Zeichen zu verarbeiten, muss es möglich sein diesen Zeichen ihre besondere Bedeutung wieder wegzunehmen. Diesen Vorgang nennt man *maskieren*.

Beispiel:

Der Text „5 ist > als 4“ soll auf dem Bildschirm ausgegeben werden.

```
$ echo 5 ist > als 4

$ ls -l
-rw-rw-r--   1 hoz      kaho            8 Aug 09 13:54 als

$ cat als
5 ist 4
```

Die Sonderbedeutung des Größerzeichens muss hier aufgehoben werden, damit es nicht als Ausgabeumlenkung aufgefasst wird. Hierzu werden Maskierungssymbole verwendet:

```
$  echo "5 ist > als 4"
5 ist > als 4

$  echo 5 ist \> als 4
5 ist > als 4
```

Folgende Maskierungssymbole können verwendet werden:

`" "` Alle Zeichen innerhalb der Anführungszeichen werden maskiert (Wirkt nicht auf die Metazeichen ` und $).

`' '` Alle Zeichen innerhalb der Hochkommata werden maskiert.

`\` Maskiert das nachfolgende Zeichen (auch `"` bzw. `'`).

4.7.1 Zusammenfassung der Sonderzeichen der Shell

Metazeichen	*Bedeutung*
n>	Ausgabeumlenkung des Kanals *n* (default: 1)
n>>	Ausgabeumlenkung (anhängen) des Kanals *n* (default: 1)
<	Eingabeumlenkung
<<	Eingabeumlenkung (Here script)

n>&*m*	Verbinden des Kanals *n* mit dem Kanal *m*
|	Pipeline
^	Pipeline (veraltetes Symbol)
* ? [..]	Jokerzeichen
"" '' \	Maskierungszeichen
`*kommando*`	Befehlssubstitution
; <NEWLINE>	Kommandoabschluß
&	Kommandoabschluß (Ausführung im Hintergrund)
#	Alles ab diesem Zeichen bis zum Ende der Zeile ist Kommentar
:	So was ähnliches wie ein Kommentarzeichen
{ *kommandoliste* }	Kommandoklammerung (Ausführung in der gleichen Shell)
(*kommandoliste*)	Kommandoklammerung (Ausführung in neuer Shell)
$*var* ${*var*}	Variablensubstitution
var=*wert*	Variablenzuweisung
kommando && *kommando* *kommando* || *kommando*	Bedingte Kommandoausführung

4.8 Hintergrundprozesse

Der Benutzer kann veranlassen, dass von ihm eingegebene Kommandos im „Hintergrund“ ablaufen. Während ein solcher Hintergrundprozess abläuft kann im Vordergrund weitergearbeitet werden.

4.8.1 Aufruf von Hintergrundprogrammen

Das Sonderzeichen um Prozesse (Kommandos) im Hintergrund zu starten ist `&`.

```
$  kommando &
```

Der Befehlsinterpreter gibt danach eine Nummer aus unter der dieser Prozess anzusprechen ist. Diese Nummer wird für jeden Prozess vergeben (auch Vordergrundprozesse erhalten eine solche Nummer) und ist im System eindeutig. Sie wird als „Process Identification“ (PID) bezeichnet.

Beispiel:

```
$  cc bsp.c &
1243
```

Die Beendigung eines Hintergrundprozesses wird nicht angezeigt.

Bei Kommandointerpretern mit Jobsteuerung wird für jeden Hintergrundprozess zusätzlich eine JobID vergeben. Bei diesen Komandointerpretern wird das Ende eines Prozesses bei dem Absetzen des nächsten Kommandos angezeigt.

4.8.2 Abbruch von Hintergrundprozessen

Hintergrundprozesse können nicht mit Hilfe der Taste `<^C>` abgebrochen werden, da sonst ein gerade laufender Vordergrundprozess mit abgebrochen würde.

Hintergrundprozesse werden mit dem `kill` Kommando abgebrochen.

kill (kill process)

Syntax:

kill [**–signalnr**] *pid*

Beispiel:

```
$  ls -lR /usr > tree &
45
$  kill 45
```

Kommandointerpreter mit Jobsteuerung erlauben auch die Eingabe der Jobid durch ein vorangestelltes Prozentzeichen anstelle der PID.

```
$  ls -lR /usr > tree &
[1] 489
$  kill %1
[1]+  Terminated ls -lR /usr > tree
```

Die `signalnr` ist eine Zahl zwischen 1 und SIGMAX die angibt aus welchem Grund der Prozess abgebrochen werden soll (default: 15). Moderne Komanndointerpreter erlauben auch die Angabe von symbolischen Namen anstelle der Signalnummer und können über die Option `-l` eine Liste aller Signalnummern, sowie der symbolischen Namen ausgeben.

```
$ kill -l | head -8
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
```

Ein Prozess hat die Möglichkeit bestimmte Signalnummern zu ignorieren.
Wird jedoch das Signal mit der Nummer 9 verwendet, wird der Prozess auf jeden Fall abgebrochen. Diese Signalnummer sollte ausschließlich verwendet werden falls der Prozess sich mit keiner anderen Prozessnummer abbrechen lässt.

Mit dem Kommando `kill` können nur eigene Prozesse abgebrochen werden (es sei denn man ist der Superuser).

4.8.3 Überwachen des Prozessstatus

Da die Beendigung von Hintergrundprozessen nicht angezeigt wird, muss es eine Möglichkeit geben, sich über die gerade laufenden Prozesse zu informieren. Das Kommando *ps*(1) unterscheidet sich grundsätzlich bei Unix SystemV und BSD kompatiblen Systemen. Im folgenden wird die SystemV Variante des Kommandos gezeigt.

ps (process status)

Syntax:

ps [**–fe**] [**–u** *user*] [**–t** *terminal*]

Wird `ps` ohne Option aufgerufen, gibt es den Status der von dem Benutzer auf dem aktuellen Terminal gestarteten Vorder– und Hintergrundprozesse aus.


```
$ ps
PID   TTY    TIME  CMD
30    ttyp1  0:10  sh
41    ttyp1  0:01  ps
|      |      |     |
|      |      |     Kommandoname
|      |      |
|      |     Zeit, die der Prozess bisher verbraucht hat
|      |
|      Terminalname
|
Prozessnummer
```

In der Ausgabe werden immer mindestens zwei Prozesse angezeigt werden: Zum einen der Kommandointerpreter (hier `sh`) und zum anderen das gerade gestartet Programm `ps` zum Anzeigen der Prozesse.

Mit den Optionen `-f` und `-e` wird eine lange Liste aller im System laufender Prozesse angezeigt.

```
$  ps -ef
     UID   PID  PPID  C    STIME  TTY  TIME COMMAND
    root     0     0  0  Mar 31   ?    0:00 sched
    root     1     0  0  Mar 31   ?    0:50 /etc/init
    root     2     0  0  Mar 31   ?    0:00 vhand
    root     3     0  0  Mar 31   ?    0:04 bdflush
    root  1382     1  0  Apr  5   01   0:00 /etc/getty tty01 sc_m
    root   539     1  0  Apr  5   02   0:00 /etc/getty tty02 sc_m
    root   153     1  0  Apr  5   ?    0:04 /etc/cron
    root   169     1  0  Apr  5   ?    0:15 tbinit
    root   159     1  0  Apr  5   ?    0:01 /usr/lib/lpsched
    root   266     1  0  Apr  5   ?    0:00 /usr/lib/lpd start
    mmdf   273     1  0  Apr  5   ?    0:00 /usr/mmdf/bin/deliver -b
    root   241     1  0  Apr  5   ?    0:00 inetd
    root  3245   241  0 10:21:49  ?    0:55 telnetd
     hoz  3246  3245  0 10:21:49  p0   0:01 -ksh
     hoz  3597  3246 13 15:32:59  p0   0:00 ps -ef
```

Die Option `-u` *name* beschränkt die Ausgabe auf die Prozesse des angegebenen Benutzers, die Option `-t` *terminal* auf das entsprechende Terminal.

4.9 Aufgaben

1. Erstellen Sie in ihrem `lib`–Verzeichnis eine Datei mit dem Namen `commands`, die eine sortierte Liste aller Unix–Kommandos aus den Verzeichnissen `/bin`, `/usr/bin` und `/usr/local/bin` enthält.
2. Wieviel Zeilen enthält die Datei `commands`?
3. Ermitteln Sie die durchschnittliche Länge der Kommandonamen.
4. Schreiben Sie eine Kommandofolge, die Ihnen nach Ablauf von 10 Minuten eine Nachricht auf den Bildschirm schreibt. Sie möchten in der Zwischenzeit normal weiterarbeiten.
5. Was bewirken die Anführungszeichen bei folgendem `find`–Befehl?
   ```
   $  find . -name "*.c" -print
   ```
6. Welche Ausgabe liefert das Kommando
   ```
   $  echo *.c
   ```
   in ihrem Loginverzeichnis und in dem Verzeichnis `src/c`?
7. Warum taucht bei
   ```
   $  ls > ls_out
   ```
   ls_out in der Liste der Dateinamen auf?
8. Erklären Sie den Unterschied zwischen
   ```
   $  who | sort
   ```
   und
   ```
   $  who > sort
   ```


5. Textverarbeitung unter Unix

Zu der Zeit als Unix entwickelt wurde, waren Textsatzsystem noch unerschwinglich und mangels graphischer Ein- und Ausgabegeräte auch immer nur als Batchsysteme ausgelegt. Der Text wurde dabei zusätzlich zum Inhalt mit Formatierungs- und Auszeichnungskommandos versehen und anschliessend mittels eines Textsatzprogramms für den Druck aufbereitet.
Unter Unix wurde damals die troff Familie entwickelt. Das Programm *nroff* war für die Formatierung von Texten und die Ausgabe auf Monospace Druckern (Typenrad) gedacht. Das Kommando *troff* produzierte eine Ausgabe für ein professionelles Satzsystem und später auch für die Ausgabe auf Postscript Druckern. Die Eingabesprache war für beide Programme identisch. Die *[tn]roff* Kommanods werden in der Regel nicht direkt genutzt sondern durch sog. Makropakete in für die Formatierung sinnvolle Kommandofolgen umgesetzt. Für troff existieren mehrere Makropakete für unterschiedliche Dokumenttypen und Anwendungsbereiche (s.u.).

Später (1977) ist dann von Donald E. Knuth TeX entwickelt worden, welches bis heute, nicht nur auf Unix Systemen, ein professionelles System zum Textsatz darstellt und sehr hochwertige Ausgaben erzeugt. Für TeX gibt es ebenfalls unterschiedliche Makropakete, die die Nutzung und Anwendung für bestimmte Aufgaben erleichtern. Das bekannteste und verbreiteste Makropaket für TeX ist LaTeX.

Inzwischen steht unter Unix das freie OfficePaket *Libre Office*, früher auch *Open Office* genannt, zur Verfügung. Damit lassen sich weitestgehend nahezu alle Microsoft Office Dokumente auch unter Unix bearbeiten. Libre Office ist allerdings, genau wie die Office Familie von Microsoft, ein eher abgeschlossenes System mit dem ein Dokumentenaustausch nur innerhalb der Office Programme möglich ist. Darüber hinaus setzen diese Programme zwingend eine graphischen Benutzeroberfläche voraus.

5.1 Groff

Heute gibt es eine Implementierung des *troff* Satzsystems unter der GNU Lizenz. Das Kommando heißt entsprechend *groff* (1). Dieses verbindet die Funktionalität der beiden originären Programme und hat einige Erweiterungen zu bieten. So werden diverse Ausgabegeräte unterstützt, wobei jedoch im wesentlichen Postscript und für Terminals die Ausgabe von Latin1 bzw. UTF8 Zeichen zu nennen wären.

Die Eingabedatei zu [ntg]roff enthält neben dem eigentlichen textlichen Inhalt zusätzliche Formatierungskommandos. Die unterstützten Kommandos hängen davon ab welches Makropaket beim Aufruf des Formatierers geladen wird. Makropakete sollen die Arbeit mit dem Satzsystem vereinfachen und bieten dem Anwendungszweck angepasste Formatierungsfunktionen und Kommandos an.
Inzwischen existieren eine Fülle von unterschiedlichen Makropaketen. Historisch wurde vor allem das *ms* (*groff_ms*(7)) und *mm* (*groff_mm*(7)) Paket für die Formatierung von mehrseitigen Dokumenten bis hin zu Büchern, sowie das *man*(7) Makropaket für die Formatierung der Manualseiten verwendet. Neuere Makropakete für die gleichen Zwecke heissen *mom* bzw. *mdoc*.

Im folgenden ist eine Eingabedatei für die Manualseite eines einfachen Unix Komandos zu sehen:


```
.TH TTY "1" "September 2007" "GNU coreutils 6.9" "User Commands"
.SH NAME
tty \- print the file name of the terminal connected to standard input
.SH SYNOPSIS
.B tty
.RI [ OPTION ]...
.SH DESCRIPTION
.PP
Print the file name of the terminal connected to standard input.
.TP
.BR \-s ", " \-\-silent ", " \-\-quiet
print nothing, only return an exit status
.TP
.B \-\-help
display this help and exit
.TP
.B \-\-version
output version information and exit
.SH AUTHOR
Written by David MacKenzie.
.SH "REPORTING BUGS"
Report bugs to <bug\-coreutils@gnu.org>.
.SH COPYRIGHT
Copyright \(co 2007 Free Software Foundation, Inc.
.br
This is free software.  You may redistribute copies of it under the terms of
the GNU General Public License <http://www.gnu.org/licenses/gpl.html>.
There is NO WARRANTY, to the extent permitted by law.
.SH "SEE ALSO"
The full documentation for
.B tty
is maintained as a Texinfo manual.  If the
.B info
and
.B tty
programs are properly installed at your site, the command
.IP
.B info tty
.PP
should give you access to the complete manual.
```

Durch die Kommandos

```
$ groff -man tty.1 | lp
$ groff -man tty.1 > tty.1.ps
```

beziehungsweise

```
$ groff -Tutf8 -man tty.1
$ groff -Tutf8 -man tty.1 | less
```

erzeugt man aus dieser Eingabedatei eine Ausgabe in Form eines Postscript Dokumentes (erste Form), oder man generiert (wie im zweiten Beispiel) eine Bildschirmausgabe für ein UTF-8 fähiges Terminal. Über die Option `-T` wird dabei das Ausgabeformat gewählt, wobei Postscript voreingestellt ist. Die Postscriptausgabe auf dem Bildschirm anzuzeigen ist wenig sinnvoll, daher sendet man diese typischerweise mit einer Pipeline zu dem Druckkommando *lp*(1) um es auf einem Drucker auszugeben, oder speichert den Inhalt mit einer Ausgabeumlenkung zur weiteren Bearbeitung in einer Datei. Die Ausgabe auf dem Bildschirm wird in der Regel über eine Pipeline zu einem Pager (*pg*(1), *more*(1) oder *less*(1)) zur seitenweisen Anzeige gesendet. Ber Postscript Ausgaben kann auch ein Postscriptbetrachter genutzt werden.

Beispielausgabe des Kommandos *groff -Tutf8 -man tty.1*

```
TTY(1)                          User Commands                          TTY(1)



NAME
       tty - print the file name of the terminal connected to standard input

SYNOPSIS
       tty [OPTION]...

DESCRIPTION
       Print the file name of the terminal connected to standard input.

       -s, --silent, --quiet
              print nothing, only return an exit status

       --help display this help and exit

       --version
              output version information and exit

AUTHOR
       Written by David MacKenzie.

REPORTING BUGS
       Report bugs to <bug-coreutils@gnu.org>.

COPYRIGHT
       Copyright © 2007 Free Software Foundation, Inc.
       This  is  free  software.   You may redistribute copies of it under the
       terms      of      the      GNU      General       Public       License
       <http://www.gnu.org/licenses/gpl.html>.   There  is NO WARRANTY, to the
       extent permitted by law.

SEE ALSO
       The full documentation for tty is maintained as a Texinfo  manual.   If
       the info and tty programs are properly installed at your site, the com-
       mand

              info tty

       should give you access to the complete manual.



GNU coreutils 6.9               September 2007                         TTY(1)
```

Beispielausgabe des Kommandos *groff -Tps -man tty.1*

TTY(1) User Commands TTY(1)

NAME

tty − print the file name of the terminal connected to standard input

SYNOPSIS

tty [*OPTION*]...

DESCRIPTION

Print the file name of the terminal connected to standard input.

−s, **−−silent**, **−−quiet**
print nothing, only return an exit status

−−help display this help and exit

−−version
output version information and exit

AUTHOR

Written by David MacKenzie.

REPORTING BUGS

Report bugs to <bug−coreutils@gnu.org>.

COPYRIGHT

SEE ALSO

The full documentation for **tty** is maintained as a Texinfo manual. If the **info** and **tty** programs are properly installed at your site, the command

info tty

should give you access to the complete manual.

GNU coreutils 6.9 September 2007 1

groff (typesetting)

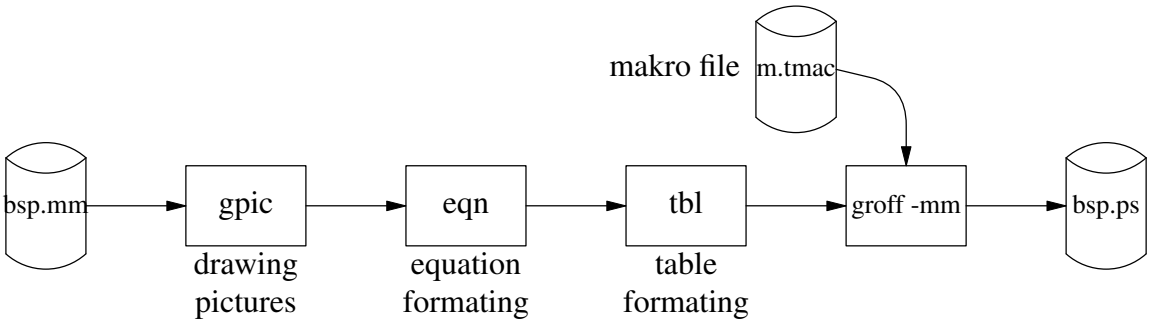
Syntax:

groff [**–egGjptsR**] [**–k**] [**–K** *enc*] [**–T** *device*] **–m** *macropkg* [*inputfile ...*]

`-egGjptsR`	Startet diverse Präprozessoren (s.u.)
`-k`	Die Eingabedatei wird durch *preconv* bearbeitet. Dies erlaubt die Nutzung von (u.a.) UTF8 Eingabedateien.
`-Kencoding`	Gibt das zu verwendende Encoding für die Eingabedateien an. Default ist die Nutzung der *locale* Umgebungsvariablen.
`-Tlatin1`	Ausgabe für Latin1 Drucker und/oder Bildschirme
`-Tutf8`	Ausgabe für UTF8 Drucker und/oder Bildschirme
`-Thtml`	Erzeugt html Dokumente als Ausgabe
`-Tlj4`	Ausgabe ist geeignet für HP Laserdrucker
`-man`	das klassiche Makropaket zur Formatierung von Unix Manualseiten
`-mdoc`	Das BSD Makropaket zur Formatierung von Unix Manualseiten
`-ms`	Einfaches Makropacket zur Formatierung von Dokumenten
`-mm`	Makropacket zur Formatierung von Dokumenten im Bell Labs Stil

5.1.1 Präprozessoren

Der Textformatierer stellt lediglich die grundlegenden Komandos für den Drucksatz zur Verfügung. Um spezielle Formatierungen durchzuführen existieren eine Fülle von Präprozessoren. Diese werden, ganz im Unix Stil, in einer Pipeline aufgerufen.




```
$ cat bsp.mm | gpic | eqn | tbl | groff -mm > bsp.ps
```

Da eine derartige Kommandokette häufig benötigt wird, kann *groff*(1) die Kommandos auch durch Optionen gesteuert aufrufen.

```
$ groff -p -e -t -mm bsp.mm > bsp.ps
```

Das Kommando *gpic*(1) stellt eine eigene Sprache zur Beschreibungen von einfachen graphischen Symbolen wie Rechteck, Kreise, Linien und Pfeile dar. Das obige Bild wurde z.B. mit *gpic*(1) erzeugt.

Der Präprozessor *eqn*(1) dient hingegen für die Beschreibung mathematischer Formeln, und *tbl*(1) wird für das Formatieren von Tabellen verwendet.

Weitere *troff* Präprozessoren sind *chem*(1) für chemische Verbindungen, *grap*(1) für das Zeichnen von statistischen Graphen, oder *refer*(1) für das Erstellen von Literaturverzeichnissen.

5.2 Markdown

Markdown ist keine Textverarbeitung im herkömmlichen Sinne. Es handelt sich dabei vielmehr um ein Programm, welches aus einfachen Textdateien HTML-Dateien erzeugt. Dabei werden bestimmte Arten der Auszeichnung von Text, in ähnlciher Form wie sich dies bereits bei (textbasierten) E-Mails durchgesetzt hat, in entsprechende HTML Anweisungen umgesetzt. Die Idee und ein *perl*(1) Programm um die HTML Dateien zu erzeugen stammt von John Gruber[7] und stellt mehr oder weniger die Referenz für eine ganze Reihe von ähnlich arbeitenden Kommandos dar. Die Vielzahl der Programme und die Tatsache das es keinen einheitlichen Standard gibt schränkt die Universalität ein wenig ein. Dies ist insbesondere deshalb schade, da ja die Nutzung von reinem Text (und auch von HTML) gerade die Portabilität erhöhen soll.

Die grundlegende Idee von *markdown* ist es keine zusätzlichen Formatierungskommandos in den Text einzustreuen[8], sondern diese aus der textuellen Darstellung zu gewinnen. Damit bleibt der Text als solcher gut lesbar wie das etwas untypische Beispiel[9], der Manualseite von *tty*(1) in Markdown[10] Syntax zeigt.

```
## NAME
tty -- print the file name of the terminal connected to standard input
## SYNOPSIS
**tty** [_OPTION_]
## DESCRIPTION
Print the file name of the terminal connected to standard input.
= `-s, --silent, --quiet` =
    print nothing, only return an exit status
= `--help` =
    display this help and exit
= `--version` =
    output version information and exit
## AUTHOR
Written by David MacKenzie.
## REPORTING BUGS
Report bugs to <bug-coreutils@gnu.org>.
## COPYRIGHT
Copyright (c) 2007 Free Software Foundation, Inc.
This is free software.  You may redistribute copies of it under the terms of
the GNU General Public License <http://www.gnu.org/licenses/gpl.html>.
There is NO WARRANTY, to the extent permitted by law.
## SEE ALSO
The full documentation for *tty* is maintained as a Texinfo manual.
If the *info* and *tty* programs are properly installed at your site,
the command `info tty` should give you access to the complete manual.
```

Wird aus diesem Text eine HTML Seite erzeugt, so werden z.B. in `'*'` eingeschlossene Worte Fett gedruckt, in Backticks geschriebene mit einem konstanten Font dargestellt und in `'_'` eingeschlossen Worte unterstrichen resp. Kursiv dargestellt. Header werden durch eine entsprechende Anzahl von #-Zeichen eingeschlossen und Aufzählungen werden eingerückt mit vorgestellten Marken (z.B. * oder −) geschrieben. Darüber hinaus gibt es Darstellungen für (HTML-)Verweise, Zitate und Code Blocks.

7. `http://daringfireball.net/projects/markdown/`
8. Diese „Auszeichnung“ von Text wird als Markup bezeichnet. Markdown soll sozusagen das Gegenteil darstellen.
9. Dafür gibt es spezialisierte Markdown Programme z.B. *ronn*.
   `https://rtomayko.github.io/ronn/ronn-format.7.html`
10. Basierend auf der in C geschriebenen Markdown Version von David Parsons
    `http://www.pell.portland.or.us/~orc/Code/discount/`

NAME

tty – print the file name of the terminal connected to standard input

SYNOPSIS

tty [*OPTION*]

DESCRIPTION

Print the file name of the terminal connected to standard input.

`-s, --silent, --quiet`
: print nothing, only return an exit status

`--help`
: display this help and exit

`--version`
: output version information and exit

AUTHOR

Written by David MacKenzie.

REPORTING BUGS

Report bugs to bug-coreutils@gnu.org.

COPYRIGHT

Copyright © 2007 Free Software Foundation, Inc.

This is free software. You may redistribute copies of it under the terms of the GNU General Public License http://www.gnu.org/licenses/gpl.html. There is NO WARRANTY, to the extent permitted by law.

SEE ALSO

The full documentation for *tty* is maintained as a Texinfo manual. If the *info* and *tty* programs are properly installed at your site, the command `info tty` should give you access to the complete manual.

Markdown wird aktuell häufig in OpenSource Projekten zum Schreiben der README Dateien verwendet (meist haben diese dann die Kennung `.md`). Aus diesen können dann automatisiert HTML Seiten generiert werden. Eine ähnliche Vorgehensweise wird auch bei Wikis gerne angewendet.
Ein gutes Beispiel und eine Referenz ist die Webseite von J. Gruber[11] sowie die inoffizielle Statusseite zu Markdown[12].

5.3 Postscript Werkzeuge

Unix Systeme bringen eine Reihe von Werkzeugen für die Bearbeitung von Postscript Dateien mit.

11. `http://daringfireball.net/projects/markdown/syntax`
12. `http://markdown.de/`

a2ps (ascii to postscript)

Syntax:

a2ps [**–E**] [**–o** *outputfile*] *textfile* [...]

`a2ps` formatiert Textdateien (z.B. SourceCode) für die Ausgabe auf Postscript Druckern.

ps2pdf (postscript to pdf konverter)

Syntax:

ps2pdf *inputfile.ps outputfile.pdf*

`ps2pdf` konvertiert Postscript- in PDF Dateien. Anstelle der Ein- und Ausgabedatei kann auch die Standardein- bzw. Ausgabe, gekennzeichnet durch ein Minuszeichen (−). angegeben werden.

psnup (postscript number of page)

Syntax:

psnup [**–1|2|3|4**] [*inputfile.ps*] [*outputfile.ps*]

`psnup` arrangiert mehrere Seiten auf einer physikalischen Seiten. Damit kann dieses Dokument z.B. zweiseitig gedruckt werden.

```
$ psnup -2 unix.ps unix-2pages.ps
```

Soll das Skript als Buch gebunden werden, sollten die Seiten vorher entsprechend angeordnet werden:

```
$ psbook unix.ps | psnup -2 > unix-2pages.ps
```

psselect (select pages from postscript)

Syntax:

psselect [**–e|o**] [**–p** *list,of,pages*] [*inputfile.ps*] [*outputfile.ps*]

Selektiert einzelne Seiten aus einem Postscript Dokument, und erstellt ein neues Postscript Dokument. Durch `-e` bzw. `-o` werden nur die geraden resp., die ungerade Seiten selektiert. Die Option `-p` erlaubt die Angabe einer expliziten Liste von Seiten

psmerge (merge several postscript files into one)

Syntax:

psmerge [**–o** *outputfile.ps*] *inputfile1.ps inputfile2.ps* [...]

Erzeugt aus allen Eingabedateien ein neues Postskript Dokument als Ausgabedatei. Die Option `-o` legt den Namen der Ausgabedatei fest. Ohne diese wird das Ergebnis auf die Standardausgabe geschrieben.

pdfunite (PDF page merger)

Syntax:

pdfunite *inputfile1.pdf inputfile2.pdf* [...] *outputfile.pdf*

Erzeugt aus allen PDF Eingabedateien ein neues PDF Dokument. Das Kommando *psmerge* kann ebenfalls benutzt werden, allerdings spart man sich die Konvertierung in Postskript falls die Eingabedateien als PDF vorliegen.

Achtung: Alle ps-Kommandos sind sehr untypische UNIX Kommandos, da sie als Argumente auf der Kommandozeile nur eine Ein- und als zweiten Dateiparameter dann eine Ausgabedatei erwarten. Dies kann ungewollt zu Datenverlust führen, wenn man die Kommandos im Zusammenhang mit Wildcards benutzt. Der Versuch mit *psnup -2 chapter[1-9].ps* alle Postscriptdateien mit dem Namen "chapter" gefolgt von einer Nummer in eine zweiseitige Form zu bekommen, führt dazu, dass das zweite Kapitel gelöscht wird, da *psnup*(1) den zweiten Parameter als Ausgabedatei für die Datei die über den ersten Parameter angegeben wurde verwendet.

gv (view postscript files)

Syntax:

gv [**−options**] [*inputfile.ps*] [...]

Ghostview ist ein graphisches Ausgabeprogramm für Postscript und PDF Dateien. Es arbeitet daher nicht auf rein textbasierten Ausgabegeräten sondern öffnet nach dem Start ein eigenes Fenster, indem die angegebene Datei angezeigt und betrachtet werden kann. Das Design des Programms ist etwas antiquiert, aber es erlaubt das Rescanning der angezeigten Datei, d.h. wenn eine neue Version (neues Dateidatum) der Postscriptdatei existiert, wird diese neu geladen und an der gleichen Stelle wie vorher dargestellt.

Zur Darstellung von PDF Dateien, insbesondere PDF Präsentationsfolien, eignen sich andere Dateibetrachter jedoch besser (s.u.).

evince (view pdf files)

Syntax:

qpdfview [*file.pdf[#page]*] [...]

Der Gnome PDF und Postscript Viewer. Verarbeitet Anmerkungen lesend und schreibend und unterstützt ein automatisches Einlesen falls sich das PDF Dokument geändert hat.

qpdfview (view pdf files)

Syntax:

qpdfview [*file.pdf[#page]*] [...]

Ein Linux Programm zur Anzeige von PDF Dateien in einem graphischen Fenster. Ein guter Ersatz für den Acrobat Reader von Adobe.

zathura (view pdf files)

Syntax:

zathura [**–P** *pagenumber*] [**–w** *password*] [*file.pdf*] [...]

Wer den Editor *vi* benutzt wird sich schnell mit *zathura* anfreunden können, wird er doch mit entsprechenden Tasten gesteuert. *zathura* erkennt Änderungen an den angezeigten Dateien und sorgt automatisch für eine aktualisierte Darstellung.

5.4 Zeichensätze und Editoreinstellungen

In den Anfangszeiten von Unix wurde lediglich der 7-Bit Ascii Zeichansatz unterstützt. Umlaute oder Sonderzeichen waren damit nicht darstellbar.
Inzwischen sind alle Unix Systeme internationalisiert, d.h. es können nationale Zeichensätze benutzt werden und die Unix-Kommandos unterstützen lokale Besonderheiten bei der Ausgabe von z.B. Datum und Uhrzeit. Besonderheiten bei der Darstellung von Geldbeträgen bzw. Dezimalzahlen und andere lokale Darstellungsoptionen werden ebenfalls berücksichtigt.

Auch wenn diese Internationalisierung bereits seit vielen Jahren existiert, war doch lange Zeit der verwendete Zeichensatz immer noch ein Problem. In Deutschland wurden, gerade auf PC-basierten Unix Systemen, häufig Microsoft Zeichensätze verwendet. Wurden diese Dateien dann auf z.B. Mac Systeme transferiert, mussten erst die Sonderzeichen auf die unter Mac verwendeten Zeichensätze angepasst werden.

Erst durch die Einführung eines weltweit universalen Zeichensatzes mit dem Namen *Unicode* ist die Verwendung von Dateien rechnerübergreifend ohne Zeichensatzkonvertierung möglich. Dies setzt allerdings voraus, dass alle beteiligten Systeme und Ein-/Ausgabegeräte tatsächlich auch Unicode verwenden.

Um die Sache nicht zu einfach werden zu lassen, können Unicode Zeichensätze unterschiedlich codiert sein. Eine verbreitete Unicode Codierung ist *UTF-8*, die unter nahezu allen Unix Systemen verbreitet ist. Eine andere, die von Microsoft Windows genutzt wird, ist *UTF-16* (Little Endian) und eine dritte Art wäre UTF-32. Bei UTF-32 werden alle Zeichen in Form von vier Byte codiert, bei UTF-16 entsprechend immer mit mindestens zwei Byte. Bei UTF-8 werden alle Ascii Zeichen von 0-127 in einem Byte codiert, und nur für andere Zeichen zwei, drei oder vier Bytefolgen zur Kodierung genutzt. Diese Art der Kodierung ist daher nicht nur meist platzsparender, sondern bietet auch noch andere Vorteile[13]. Die Bytefolgen bei Multibyte UTF-8 Zeichen beginnen immer mit einem Byte deren linken Bits `11` sind (genauer: `110` bei einer Folge aus zwei Byte, `1110` bei einer Folge aus drei Byte und `11110` bei einer Folge aus vier Byte), wohingegen bei den folgenden Bytes die beiden linken Bits auf `10` gesetzt sind. Alle anderen Bits werden zur Codierung des Zeichens selber genutzt, d.h. dafür stehen dann bis zu 21 Bits zur Verfügung.

5.4.1 Zeichensatz Konvertierung

Um eine Textdatei von einem Zeichensatz resp. einer Kodierung in einen anderen Zeichensatz zu konvertieren kann das Komando *iconv*(1) benutzt werden.

13. Siehe dazu auch die Erläuterungen auf Wikipedia (https://de.wikipedia.org/wiki/UTF-8)

iconv (convert encoding of files)

Syntax

iconv –f *from-code* [**–t** *to-code*] [**–o** *outputfile*] [*inputfile ...*]

oder

iconv –l

Das Kommando konvertiert die Eingabedateien in den lokalen Zeichensatz (oder den durch die Option `-t` angegebenen). Der Zeichensatz der Eingabedatei wird durch die Option `-f` bestimmt. Die zweite Form des Kommandos listet alle Zeichensätze auf die bei der From- und To- Angabe genutzt werden können.

Der lokal aktuell genutzte Zeichensatz wird duch die Einstellungen des Kommandos *locale*(1) angezeigt und bestimmt.

Beispiel:

```
$  locale
LANG=en_US.UTF-8
LANGUAGE=en_US
LC_CTYPE=de_DE.UTF-8
LC_NUMERIC=de_DE.UTF-8
LC_TIME=de_DE.UTF-8
LC_COLLATE=C
LC_MONETARY=de_DE.UTF-8
LC_MESSAGES="en_US.UTF-8"
LC_PAPER=de_DE.UTF-8
LC_NAME=de_DE.UTF-8
LC_ADDRESS=de_DE.UTF-8
LC_TELEPHONE=de_DE.UTF-8
LC_MEASUREMENT=de_DE.UTF-8
LC_IDENTIFICATION=de_DE.UTF-8
LC_ALL=
```

5.4.2 Editor Einstellungen

Der Editor *vi*(1) bzw. sein Nachfolger *vim*(1) unterstützt ebenfalls die Konvertierung resp. die Bearbeitung von Dateien mit unterschiedlichen Zeichensätzen.

Für das Erstellen von Textdokumenten gibt es sinnvolle Einstellungen und Editierkommandos. So kann z.B. ein Abschnitt im Editor neu „formatiert“ werden indem Zeilen, die länger sind als 80 Zeichen neu umgebrochen werden. Das Formatierungskommando in *vim*lautet `gq`. Sinnvoll ist diese Art der Formatierung z.B. beim Erstellen von *Markdown* Texten. Für *groff* formatierte Texte is dies ungeeignet, da die Formatierungsanweisungen nicht erkannt werden und dadurch ebenfalls umgebrochen werden.

5.5 Versionsverwaltung

Programme zur Versionsverwaltung sind ursprünglich primär für die Verwaltung von Quelltexten bei der Programmierung entwickelt worden. Sie eignen sich allerdings sehr gut für die Verwaltung jeglicher (Text)Dateien, und damit natürlich auch für Textverarbeitung, Konfigurationsdateien, Webseiten und anderes.

Unter Unix gab es bereits seit Anfang der 1970er Jahre unter dem Namen *SCCS* ein Versionsverwaltungssystem. Später wurde dies durch leistungsfähigere Nachfolger wie *RCS*, *CVS* und aktueller dann durch *Subversion* ersetzt. Moderne Versionsverwaltungen sind *Mercury* bzw. *GIT*. Letzteres setzt sich gerade insbesondere bei Unix Anwendern im großen Stil durch. Zum Einen weil es von Linus Torvald entwickelt wurde um den Quelltext des Linux Kernels zu verwalten, und zum Anderen weil sich um diese Versionsverwaltung auch eine Plattform[14] zum Austausch von Open Source Projekten

gebildet hat.

In der Windows und Mac OS/X Welt findet *GIT* allerdings ebenfalls immer mehr Anhänger. Hier meist in Form eines Programmes mit graphischer Oberfläche. Unter Unix wird *git*(1) traditionell meist in der Kommandozeilenversion genutzt.
Die Bedienung und das Konzept von *git* ist nicht in zwei Sätzen erläutert. Hier sollen daher nur die drei wichtigsten Funktionen gezeigt werden.

1. Anlegen eines Git Repositories in dem Verzeichnis indem die zu verwaltenden Dateien stehen
   ```
   $ git init
   ```
2. Hinzufügen aller Dateien des aktuellen Verzeichnisses und aller Unterverzeichnisse in den sog. Staging Bereich:
   ```
   $ git add .
   ```
   Nicht zu beachtende Dateien können vorab über Einträge in der Datei `.gitignore` ausgeschlossen werden.
3. Ablegen der Dateien des Staging Bereichs im Repository (commit):
   ```
   $ git commit -m "Initial commit"
   ```

Weitere Git-Kommandos sind

`git status`
: Zeigt den Zustand des Arbeitsbereichs an (Alle Veränderungen commited? Was steht im Staging Bereich? etc.)

`git log`
: Anzeige von alten Revisionsständen mit der Erläuterung

`git diff [rev1 [rev2]]`
: Zeigt die Unterschiede zwischen verschiedene Revisionsständen an.

`git push/pull`
: Synchronisation des lokalen Repository mit einem remote Master. Die Kommunikation mit dem Master findet im Allgemeinen über *ssh*(1) statt.

`git branch`
: Anlegen und Löschen eines Branches

`git checkout`
: Wechsel zwischen Branches

`git merge`
: Zusammenführen zweier Branches/Versionen

14. `https://github.com`

6. Dateibearbeitung

6.1 Inhalte von Dateien anzeigen

cat (concatenate files)

Syntax:

cat [**–vte**] [**–bns**] [*datei ...*]

Schreibt die angegebenen Dateien (oder die Standardeingabe) hintereinander auf die Standardausgabe und führt dadurch eine Verkettung der Dateien durch.

Beispiel:
Aneinanderhängen aller C–Quelldateien eines Verzeichnisses in einer großen Datei.

```
$  cat *.c > cfiles
```

Wird anstelle eines Dateinamens ein Minuszeichen angegeben, wird die Standardeingabe an dieser Stelle gelesen.

Beispiel:
Ausgabe von *datei_a* und *datei_b*. Zwischen beiden werden *-chen ausgegeben.

```
$  echo "****" | cat datei_a - datei_b
```

Optionen:

`-v` (visible)	Stellt nichtdarstellbare Zeichen in der Form `^A` (für das ASCII-Zeichen 1) dar. Zeichen oberhalb 127 werden in der Form `M-x` dargestellt, wobei *x* das entsprechende Zeichen im unteren Ascii–Zeichensatz darstellt.
`-t` (tabulator)	Nur in Verbindung mit der Option `-v`. Es werden auch Tabulatorzeichen in der Form `^I` und Formfeedzeichen als `^L` dargestellt.
`-e` (end of line)	Nur in Verbindung mit der Option `-v`. Es wird vor jedem Zeilenendezeichen das Zeichen "$" ausgegeben.
`-n` (number lines)	Stellt jeder Zeile eine Zeilennummer voran.
`-b` (number non blank lines)	Stellt jeder nicht Leerzeile eine Zeilennummer voran.
`-s` (squeeze blank lines)	Mehrere aufeinanderfolgende Leerzeilen werden zu einer Leerzeile zusammengefasst.

6.1.1 Seitenweises Betrachten von Dateien

pg (outputs a page at a time)

Syntax:

pg [*Optionen*] [*datei ...*]

Zeigt den Inhalt der angegebenen Dateien (oder die Standardeingabe) seitenweise auf dem Bildschirm an.

Nach jeweils einer Seite wartet `pg` auf die Eingabe eines der folgenden Kommandozeichen.

Kommandozeichen:

<RETURN>	Ausgabe der nächsten Seite.
$	Zeigt die letzte Seite an.
n	Die Ausgabe wird bei der *n*-ten Seite fortgesetzt.
+*n*	'*n*' Seiten weiterblättern (default: Eine Seite).
−*n*	'*n*' Seiten zurückblättern (default: Eine Seite).
.	Erneute Ausgabe der aktuellen Seite.
/*muster*/	Sucht das angegebene Muster vorwärts (nach der aktuellen Seite).
?*muster*?	Sucht das angegebene Muster rückwärts (vor der aktuellen Seite).
*i*n	*i*-te nächste Datei anzeigen (default: i = 1).
*i*p	*i*-te vorhergehende Datei anzeigen (default: i = 1).
*i*w	Die nächste auszugebende Seite ist '*i*' Zeilen groß.
h	Gibt eine Hilfeseite aus.
Q oder q	Beendet pg.
!*Kommando*	Führt das angegebene Kommando aus, ohne pg zu verlassen.

Beschreibung einiger Optionen:

−c	Löscht vor der Ausgabe einer Seite den Bildschirm.
−n	Kommandos müsssen nicht mit Newline bestätigt werden.
−*k*	Die Seitenlänge ist *k*-Zeilen.
+*k*	Die Ausgabe beginnt bei Seite *k*.
+/*muster*/	Die Ausgabe beginnt mit der Zeile, in der das Muster erstmalig auftritt.
−f	Lange Zeilen werden nicht von pg umgebrochen.
−s	Promptzeichen und Meldungen werden hervorgehoben.
−p*text*	*Text* wird von pg als Promptzeichen benutzt. Die Zeichenfolge "%d" wird dabei durch die aktuelle Seitenzahl ersetzt.

pg ist ein sehr leistungsfähiges und vielseitiges Kommando. Allerdings besitzt es ein paar kleine „Merkwürdigkeiten“:

- Die Suche eines Musters beginnt immer außerhalb der aktuellen Seite, d.h. pg findet das Muster nicht, wenn es bereits auf dem Bildschirm steht.
- pg kann zwar seitenweise vorwärts und rückwärts Blättern, aber nicht zeilenweise vorwärts positionieren.
- Die Optionen von pg können nicht fest konfiguriert werden. Ich möchte z.B. das die Optionen −scnp"Seite %d: " immer gesetzt sind. Dies geht nur über eine Batchdatei.

Die oben genannten Nachteile besitzt das als Public Domain verfügbare Programm less nicht.

more (more pages)

Syntax:

more [*Optionen*] [*datei ...*]

more ist ein Pager ähnlich pg der von BSD−Unix übernommen wurde. Er hat ein paar Vorteile gegenüber pg aber einen gravierenden Nachteil:

Mit more kann man in einem Text nicht rückwärtsblättern.

less (less is more than pg)

Syntax:

less [*Optionen*] *datei* [...]

Dieser Pager ist ein Public Domain Programm das unter verschiedenen Betriebsystemen zur Verfügung steht (z.B. DOS[15], OS/2).

Der Autor scheint ein bescheidener Mensch zu sein, da `less` der Pager mit den umfangreichsten Kommandos und Konfigurationsmöglichkeiten ist.

less(1) verbindet die Vorteile von `pg` und `more`, und ist darüber hinaus noch leicht zu erlernen, da es sich an die Editierkommandos des `vi` anlehnt. Wer den `vi` bedienen kann, dürfte von daher wenig Schwierigkeiten mit der Bedienung von `less` haben.

Genauere Informationen finden sich in der Manualseite bzw. in der Onlinehilfe (Kommando: `h`).

6.2 Dateien für die Ausgabe formatieren

fold (fold lines)

Syntax:

fold [**−w** *col*] [**−b**] [**−s**] [*datei ...*]

Mit *fold*(1) werden lange Zeilen in der Eingabe nach 80 Zeichen oder bei der durch die Option −w angegeben Anzahl Zeichen umgebrochen. Tabulatoren werden auf die nächste 8er Position erweitert. Durch die Option −b wird dies verhindert. Die Option −s bricht die Zeile an der letzten Leerstelle (Space) um.

pr (print files)

Syntax:

pr [*Optionen*] [*datei ...*]

pr(1) erlaubt eine einfache Formatierung (Seiteneinteilung, linker Rand, Zeilennummerierung und mehrspaltiges Format) von Textdateien.

Wird `pr` ohne Optionen aufgerufen, gibt es die Eingabedateien seitenweise formatiert mit Kopf- und Fußzeilen auf der Standardausgabe aus. Die Kopfzeile enthält das aktuelle Datum, den Dateinamen und eine Seitenzahl.
`pr` wird in der Regel als Filterprogramm vor der Ausgabe auf dem Drucker verwendet.

Optionsbeschreibung:

+*k*	Die Ausgabe beginnt erst ab Seite *k*.
−*zahl*	Mehrspaltige Ausgabe (*zahl* gibt die Anzahl der Spalten an).
`-d`	(double space) In der Ausgabe wird nach jeder Zeile eine Leerzeile eingefügt.
`-f`	(formfeed) Am Seitenende werden Formfeedzeichen (`^L`) anstelle einer Folge von Newlines verwendet.
`-l`*len*	(lines) Die Seitenlänge beträgt *len* Zeilen. (Standard: 66; In Europa ist das Endlospapier jedoch 72 Zeilen lang)

15. Dann kann man endlich `more.exe` verbrennen.

`-o`*k* (offset)
Jede Zeile wird um *k* Zeichen vom linken Rand eingerückt. (Standard: 0)

`-h` *Kopfzeile* (head)
In der Kopfzeile soll anstelle des Dateinamens die angegebene Zeichenkette ausgegeben werden. (Die Angabe kann vor jeder auszugebenden Datei erneut erfolgen.)

`-t` (no titel)
Die 5-zeiligen Kopf- und Fußbereiche werden nicht ausgegeben.

`-n`*ck* (number)
Jeder Zeile wird eine *k*-stellige Zeilennummer vorangestellt (Standard: 5). Das optionale Zeichen *c* gibt das Trennzeichen zwischen Zeilennumer und dem Zeilentext an (default: Tabulator).

`-w`*m* (width)
Setzt die Ausgabebreite auf *m* (default: 72) Zeichen (nur bei mehrspaltiger Ausgabe).

`-e`*ck* (expand)
Expandiert Tabulatorzeichen zu *k* Leerzeichen. Wird die Angabe von *k* weggelassen, wird zu 8 Leerzeichen expandiert (ersetzt das BSD Kommando `expand`).
Das optionale Zeichen 'c' bezeichnet das Zeichen der Eingabe, welches in der Ausgabe durch Leerzeichen ersetzt werden soll (default: <TAB>).

Beispiel:

```
$  pr -t -e:12 lib/ulist
```

`-i`*cn* Die Umkehrung der Option `-e`: Leerzeichen in der Eingabe werden durch Tabulatorzeichen in der Ausgabe ersetzt.

Beispiele:

1. Die Ausgabe der Dateien des Verzeichnisses `/bin` soll in 4 Spalten erfolgen:
```
$ ls /bin | pr -3 -t -l1 | pg
$ ls /bin | pr -4 -o5 -l72 -h "Dateien aus /bin" | lp
```
2. Es sollen alle C-Quelltextdateien formatiert mit Zeilennummern auf dem Drucker ausgegeben werden. Tabulatorzeichen sollen dabei auf 4-er Positionen expandiert werden:
```
$ pr -l72 -e4 -n *.c | lp
```

6.3 Filterprogramme zur Bearbeitung von Datenströmen

tr (translate characters)

Syntax:

tr [**-cds**] [*zeichenmenge1*] [*zeichenmenge2*]

`tr` liest von der Standardeingabe und schreibt auf die Standardausgabe.

Dabei werden Zeichen, die in *zeichenmenge1* (z1) angegeben sind in die entsprechenden Zeichen der *zeichenmenge2* (z2) übersetzt. Das erste Zeichen aus *z1* wird in das erste Zeichen aus *z2* übersetzt, das zweite Zeichen in das zweite Zeichen aus *z2* usw.

Ist *z2* kürzer als *z1*, so kann durch die Angabe `[z*]` erreicht werden, dass *z2* implizit mit dem Zeichen *z* auf die Länge der *zeichenmenge1* aufgefüllt wird.

In den Zeichenmengen können nicht darstellbare Zeichen durch "\" gefolgt von der Nummer des Zeichens in oktaler Schreibweise dargestellt werden.

Zeichenbereiche können, in eckige Klammern eingeschlossen, als von..bis Angabe dargestellt werden. (Da die eckige Klammern eine Sonderbedeutung in der Shell haben, sind sie zu maskieren.)

Beispiele:

```
$  banner Text | tr "#" "*"

$ tr "a-z" "A-Z" < bsp.txt
→   ersetzt alle Kleinbuchstaben aus der Datei "bsp.txt" in
    der Ausgabe durch Großbuchstaben

$ tr ":" "\011" < lib/ulist
→   ersetzt jeden Doppelpunkt in der Datei ulist in
    der Ausgabe durch ein Tabulatorzeichen
```

Optionen:

`-c` (complement)	Komplementiert die Zeichen aus *zeichenmenge1* bezüglich des ASCII–Zeichensatzes (Negation der Zeichenmenge)
`-d` (delete)	Löscht jedes Auftreten von Zeichen aus *zeichenmenge1*
`-s` (squeeze)	Mehrfach auftretende Ausgabezeichen aus *z1* werden durch ein einzelnes Zeichen ersetzt

Beispiele:

```
$  tr -s " " "\011"    # ersetzt jede Folge von Blanks durch ein <TAB>

$  tr -cd "\040-\0177" # löscht alle nicht darstellbaren Zeichen

$  tr -sc "A-Za-z" "\012*" <bsp.txt
   # ersetzt jedes Zeichen, das kein Buchstabe ist, durch das
   # Zeilenvorschubzeichen (\012 → <NEWLINE>)
   # Jedes Wort (jede Buchstabenfolge) der Datei "bsp.txt"
   # wird auf eine extra Zeile geschrieben.
```

Neuere Versionen des Kommandos unterstützen auch die Angabe von Zeichenmengen in symbolischer Form. Möchte man z.B. alle Groß- in Kleinbuchstaben wandeln, kann dies auch durch

```
tr "[:upper:]" "[:lower:]"
```

erreicht werden. Durch die symboilische Form werden auch z.B. Umlaute korrekt umgewandelt. Ziffern und bestimmte Sonderzeichenmengen sind ebenfalls über solche symbolischen Zeichenmengen darstellbar.

Einige ASCII–Zeichen in oktaler Schreibweise:

Ascii	*Oktal*	*Esc*	*Bedeutung*	*Ascii*	*Oktal*	*Esc*	Bedeutung
`^G`	07	`\a`	Glocke	`^J`	012	`\n`	Newline
`^H`	010	`\b`	Backspace	`^L`	014	`\f`	Formfeed
`^I`	011	`\t`	Tabulator	`^M`	015	`\r`	Carriage Return

Eine Liste einiger symbolischen Zeichenmengen:

Charset	*Bedeutung*
`[:upper:]`	Großbuchstaben
`[:lower:]`	Kleinbuchstaben
`[:alpha:]`	Buchstaben
`[:digit:]`	Ziffernzeichen
`[:xdigit:]`	Hexadezimale Ziffernzeichen
`[:alnum:]`	Buchstaben und Ziffernzeichen
`[:punct:]`	Satzzeichen
`[:space:]`	Horizontale und vertikale Leerzeichen

tail (show tail of file)

Syntax:

tail [**–f**] [**+|–***n*] [*datei*]

Das Kommando schreibt die letzen 10 Zeilen einer Datei (oder der Standardeingabe) auf die Standardausgabe.

Durch die Option *–n* können die letzten *n* Zeilen der Datei dargestellt werden.
Die Angabe *+n* bedeutet, dass alle Zeilen ab der *n*-ten Zeile ausgegeben werden.

Bei *n* kann zusätzlich eine Einheit angegeben werden. (Standard: lines)

`l` (lines)	Die Angabe bezieht sich auf Zeilen.
`b` (block)	Die Angabe bezieht sich auf Blöcke (512 Byte).
`c` (char)	Die Angabe bezieht sich auf Zeichen.

Wird die Option `–f` angegeben, terminiert `tail` nicht, sondern testet in regelmäßigen Abständen ob die Datei länger geworden ist. Wenn dies der Fall ist werden die neu hinzugekommenen Daten ausgegeben. Das Kommando muss in diesem Fall explizit abgebrochen werden `<^C>`.

Beispiele:

```
# gibt den Namen der neuesten Datei des akt. Verz. aus
$ ls -tr | tail -1
$ who am i | tail -6c  # gibt die Zeit des Einloggens aus
```

Wird mehr als eine Datei angegeben, wird von jeder Datei die letzen *n* Zeilen ausgegeben. In diesem Fall wird vor jeder jeder Ausgabe der Name der Datei eingefasst in `==> file <==` vorangestellt und zwischen jeder neuen Datei eine Leerzeile eingefügt:

```
$ tail -n1 /etc/passwd /etc/group
==> /etc/passwd <==
ntp:x:117:125::/home/ntp:/bin/false

==> /etc/group <==
dnsadmin:x:99:hoz
```

head (print head of file)

Syntax:

head [**–***n*] [*file ...*]

`head` gibt die ersten *n* Zeilen (default: 10) der angegebenen Dateien (oder der Standardeingabe) auf der Standardausgabe aus.

Wird mehr als eine Datei angegeben wird die Ausgabe entsprechend des Kommandos `tail` dargestellt:

```
$ head -1 /etc/passwd /etc/group
==> /etc/passwd <==
root:x:0:0:root:/root:/bin/bash

==> /etc/group <==
root:x:0:
```

Die Kombination von `head` und `tail` erlaubt auch die Ausgabe von Zeilen in der Mitte einer Datei (Das gleiche könnte jedoch auch mit dem Kommando *sed*(1) erreicht werden):

```
$ head -n 20 /etc/passwd | tail -1
$ sed -n 20p /etc/passwd
```

uniq (list uniq lines)

Syntax:

uniq [**-d|u|c**] [*inputfile [outputfile]*]

Das Kommando reduziert aufeinanderfolgende gleiche Zeilen in der Eingabe auf eine Zeile in der Ausgabe.

ACHTUNG:
Bei mehr als einer Dateiangabe ist die zweite Datei keine Eingabedatei. (Widerspricht dem Konzept von UNIX!! Sollte man generell nicht benutzen. Vorsicht bei Wildcards!!!!)[16]

Optionen:

`-d` (duplicate)	Es werden nur die Duplikate (einmal) angezeigt.
`-u` (unique)	Es werden nur die Unikate (einmal) angezeigt.
`-c` (count)	Die Anzahl des mehrfachen Auftretens wird ermittelt und vor die Zeile geschrieben.

Beispiel:
Die Datei "bsp" enthalte eine Namensliste:

```
$ cat bsp
Hans
Hans-Erich
Hans-Erich
Hugo

$ uniq bsp
Hans
Hans-Erich
Hugo

$ uniq -d bsp
Hans-Erich

$ uniq -u bsp
Hans
Hugo

$ uniq -c bsp
   1 Hans
   2 Hans-Erich
   1 Hugo
```

16. Der Programmierer dieses Kommandos war entweder besoffen, oder hatte keine Ahnung von Unix.

cut (cut out fields)

Syntax:

cut **–f** *liste* [**–d** *trennzeichen*] [*file ...*]

oder

cut **–c** *liste* [*file ...*]

Das Kommando schneidet Felder bzw. Spalten aus Dateien oder der Standardeingabe aus und gibt diese auf der Standardausgabe aus. Die Option `-f` (**f**ields) wählt die Felder aus die ausgeschnitten werden sollen. Die Felder werden von 1 beginnend durchnummeriert.

Die Option `-d` (**d**elimiter) gibt dabei an welches Zeichen als Trennzeichen zwischen den Feldern verwendet werden soll (default: `<TAB>`).

Die Option `-c` (**c**olumns) wählt Spalten aus, die ausgeschnitten werden sollen. Die Spalten werden von 1 beginnend gezählt.

Die Liste der Optionen `-c` bzw. `-f` ist eine durch Kommata getrennte Aufzählung von Werten. Bereiche werden durch einen Bindestrich gekennzeichnet.

Beispiele:

```
$  cut -d: -f1,5 /etc/passwd          # Liste aller Benutzer mit ihrem Namen

$  who | cut -c1-11,25-
hoz         Apr 12 09:54
hoz         Apr 12 10:44
ben110      Apr 11 15:30

$  who am i | cut -c1-11          # (Fast) Das gleiche wie logname
hoz
$  who am i | cut -d" " -f1      # Was ist der Unterschied zu oben ?
```

6.3.3 Aufgaben

1. Ermitteln Sie die Namen der Benutzer die mehrfach angemeldet sind.
2. Die Datei `~/.ssh/known_hosts` enthält die öffentlichen Schlüssel der Server die per *ssh*(1) kontaktiert wurden. Die Datei enthält sehr lange Zeilen mit dem Servernamen (optional der IP-Adresse), dem Schlüssel Algorithmus und dem öffentlichen Schlüssel.
   Geben sie die Datei formatiert auf dem Bildschirm aus, sodass jeweils eine Zeile den Servernamen, eine Zeile den Schlüsselalgorithmus und in den folgenden Zeilen der öffentlichen Schlüssel dargestellt wird.

   Zusatzaufgabe: Die Einträge jeden Servers sollen durch jeweils eine Leerzeile getrennt sein.[17]

6.4 Sortieren von Dateien

sort (sort or merge lines of files)

Syntax:

sort [**–muc**] [**–t** *trennzeichen*] [*sortflag*] [*pos*] [**–o** *outputfile*] [*file ...*]

Sortiert die angegebenen Dateien (oder die Standardeingabe) und schreibt das Ergebnis auf die Standardausgabe oder in die mit der Option `-o` (**o**utput) angegebene Datei (Diese kann auch eine Eingabedatei sein).

17. Tipp: Sehen sie sich hierzu die Manualseite des Kommandos *sed*(1) an.

Als Sortierkriterium wird standardmäßig die gesamte Eingabezeile benutzt.

Sortiert wird nach dem Maschinenzeichensatz. Bei neueren Unix–Versionen (SysV.3) kann die Sortierreihenfolge an nationale Gegebenheiten angepaßt werden. Eine Umgebungsvariable (`LC_COLLATE`) stellt die gewünschte Art der Sortierung ein.

Werden mehrere Eingabedateien angegeben, wird jede einzelne Datei sortiert und anschließend die Dateien gemischt auf die Standardausgabe geschrieben, so dass die Ausgabe insgesamt sortiert ist. Wird die Option `-m` (**m**erge) angegeben werden die Eingabedateien nur gemischt. Die Eingabedateien müssen hierzu bereits sortiert vorliegen.

Bei der Option `-u` (**u**niq) werden gleiche Eingabezeilen nur einmal in die Ausgabe geschrieben.

Die Option `-c` (**c**heck) testet, ob die Eingabedateien sortiert vorliegen. (Exitstatuts: 0 wenn sortiert; ungleich 0 wenn nicht)

Beispiele:

```
$  cut -d: -f2 lib/books |      # Extrahieren der Autoren
>  tr "," "\012" |           # Mehrere Autoren untereinanderschreiben
>  sort -u                   # Sortieren und doppelte entfernen

$ ls /bin > a
$ ls /usr/bin > b
$ sort -m a b          # Die sortierten Dateien mischen
```

Jede Eingabezeile wird als Datensatz betrachtet, wobei die einzelnen Felder des Datensatzes durch *white space* voneinander getrennt sind.

Durch die Angabe der Option `-t`*x* (**t**rennzeichen) wird das Zeichen *x* als Trennzeichen zwischen den Feldern verwendet. (Wenn *x* ein Sonderzeichen für die Shell ist, muss es maskiert werden.)

Beispiel:
Die Datei `lit/books` enthält Zeilen mit folgendem Aufbau:

Buchnr:Autor:Verlag:Jahr:Titel:Kurzbezeichnung

Jede Zeile der Datei enhält einzelne Felder, die durch Doppelpunkt voneinander getrennt sind. Bei der Anwendung von `sort` auf diese Datei, sollte man als Feldtrennzeichen `-t:` angeben.

```
$ sort -t: lit/books
```

`sort` nummeriert die einzelnen Felder einer Zeile mit 0 beginnend. Diese Feldnummern können in Positionsangaben verwendet werden.

Buchnr	Autor	Verlag	Jahr	Titel	Kurzbezeichnung
0	1	2	3	4	5

6.4.1 Positionsangaben

Wenn als Sortierkriterium nicht die gesamte Eingabezeile dienen soll, muss dies durch Positionsangaben kenntlich gemacht werden.

Werden mehrere Positionsangaben gemacht, bezieht sich die erste auf das Hauptsortierkriterium, die zweite auf das nächste Sortierkriterium usw.

Eine Positionsangabe besteht aus den Angaben:

`+`*n* `[-`*m*`]`

Dabei sind *n* und *m* Positionsangaben, die den Bereich kennzeichnen, der als Sortierkriterium dienen soll (von Feldnummer *n* bis vor Feldnummer *m*).

Wird die Angabe –*m* weggelassen, gilt das Sortierkriterium bis zum Zeilenende.

Als weitere Einschränkung des Sortierbegriffs kann die Position in einem Feld durch die Angabe eines Punktes, gefolgt von einer Zahl, hinter der Feldnummer erfolgen.
Die Position innerhalb eines Feldes wird mit 0 beginnend gezählt.

Beispiel:
Sortieren der Datei "books", wobei als erstes nach Verlagen und bei gleichen Verlagen nach Autoren sotiert werden soll.

```
$ sort -t: +2 -3 +1 -2 lib/books
```

Beispiel:
Sortieren nach Buchnummern, wobei der Buchtyp nicht beachtet werden soll. (So nicht ganz korrekt).

```
$ sort -t: +0.2 -1 books
```

6.4.2 Sortierflags

Sortierflags geben an wie das entsprechende Feld sortiert werden soll. Sie können als Einzeloption angegeben oder an eine Positionsangabe angehängt werden.

Bei der Angabe als Einzeloption bezieht sich das Flag auf alle nachfolgenden Felder.

Als Sortierflag sind folgende Angaben (auch mehrere) erlaubt:

`f` (**f**old away)	Beim Sortieren wird nicht zwischen Groß- und Kleinbuchstaben unterschieden.
`i` (**i**gnore)	Beim nichtnumerischen Vergleich werden nichtdarstellbare Zeichen ignoriert. Welche Zeichen als nichtdarstellbar gelten, wird durch lokale Einstellungen bestimmt.
`r` (**r**everse)	Die Sortierreihenfolge wird umgekehrt (Standard: aufsteigend).
`b` (**b**lanks)	Beim Sortieren sollen führende Blanks ignoriert werden (Nur im Zusammenhang mit Positionsangaben wirksam).
`M` (**M**onth)	Die ersten 3 Stellen des Feldes werden als Monatsangabe der Form "Jan", "Feb" usw. aufgefaßt (`ls -l` liefert eine solche Form).
`d` (**d**ictionary)	Beim Vergleich werden nur Buchstaben und Ziffern verwendet.
`n` (**n**umeric)	Das Feld soll numerisch sortiert werden.

Beispiele:

```
$ sort -t: +0.2nr -1 books # Sortiert nach Buchnummern absteigend

$ ls -l /etc | sort +5M -6 +6n -7 +8r
#   Sortiert die Dateien aus "/etc" nach Monat und Tag; Bei
#   gleichem Datum absteigend nach den Dateinamen

$ sort -t: +3n -4 +0r -1 /etc/passwd
#   Sortiert die Datei /etc/passwd nach der Guppennummer und innerhalb
#   einer Gruppe nach den Benutzernamen.
```

Ermitteln der Häufigkeit der einzelnen Worte in den Dateien mit der Kennung ’txt’, absteigend sortiert nach der Häufigkeit.

```
$ cat *.txt |          # Alle Dateien mit der Endung txt
> tr -sc "[A-Z]ÄÖÜ[a-z]ßäöü" "[\012*]" |  # in Worte aufspalten (Zeilenweise)
> tr "[A-Z]ÄÖÜ" "[a-z]äöü"  |  # Groß- in Kleinbuchstaben konvertieren
> sort |          # Sortieren der Eingabe
> uniq -c |           # gleiche Zeilen zählen
> sort -rn            # nach Anzahl sortieren
```


6.4.3 Aufgaben

1. Erstellen Sie eine Liste aller Benutzergruppen mit der Anzahl ihrer Mitglieder (siehe `/etc/passwd`). Die Ausgabe soll nicht den Gruppennamen sondern die Gruppennummer beinhalten.
2. Wie kann man die Benutzergruppe mit den meisten Mitgliedern ermitteln.

6.5 Suchen von Textmustern in Dateien

Die "grep" Familie:

`grep`	**g**lobally **r**egular **e**xpression **p**rint
`fgrep`	**f**ile **grep**
`egrep`	**e**xtended **grep**

Die `grep` Programme durchsuchen Dateien (oder die Standardeingabe) nach einem Suchmuster und geben jede Zeile die das Suchmuster enthält auf dem Bildschirm aus.

Wenn mehr als eine Datei angegeben ist, wird zusätzlich der Dateiname angezeigt.

Die Programme unterscheiden sich durch die Möglichkeiten bei der Suchmusterangabe:

grep	Als Muster sind eingeschränkte reguläre Ausdrücke erlaubt.
fgrep	Als Suchbegriff können nur konstante Zeichenketten angegeben werden. Dafür können aber sehr viele Suchbegriffe angegeben werden. Diese müssen dann in einer Datei abgelegt sein (siehe Option `-f`).
egrep	Als Muster sind erweiterte reguläre Ausdrücke erlaubt.

Reguläre Ausdrücke werden im nächsten Abschnitt erläutert.

Die allgemeine Syntax der Kommandos ist im wesentlichen gleich.

Syntax:

[ef]grep [*optionen*] *suchmuster* [*file ...*]

Das Suchmuster sollte in Hochkommata eingeschlossen werden, um Zeichen mit einer Sonderbedeutung zu maskieren.

Als Optionen sind möglich:

`-v`	(in**v**ert)	Nur Zeilen ausgegeben, auf die das Muster **nicht** passt (default: Passende Zeilen).
`-c`	(**c**ount)	Nur die Anzahl der passenden Zeilen ausgegeben.
`-l`		Gibt ausschließlich den Dateinamen aus, nicht die Zeile mit dem Suchbegriff.
`-n`	(**n**umber)	Gibt die Zeilennumer vor jeder Zeile aus.
`-s`	(**s**ilent)	Keine Fehlermeldungen bei nicht lesbaren Dateien usw.
`-h`		Die Dateinamen werden nicht mit ausgeben.
`-H`		Die Dateinamen werden immer ausgeben, nicht nur bei mehreren Eingabedateien.
`-i`	(**i**gnore)	Beim Vergleich soll Groß- und Kleinschreibung ignoriert werden.
`-w`	(**w**ord)	Das Muster soll als alleinstehendes Wort gesucht werden.
`-e`*muster*	(**e**xpression)	Sinnvoll, falls das Muster mit "-" beginnt oder mehrere Muster angegeben werden.

`-f`*datei*	(**f**ile)	Das Muster nach dem gesucht werden soll, steht in der angegebenen Datei. Die Datei darf mehrere Suchbegriffe enthalten, die jeweils in einer neuen Zeile stehen müssen.
`-r`	(**r**ecursiv)	Rekursive Suche

Heutige Versionen der *grep*(1) Kommandos besitzen die Möglichkeit einige Zeilen vor und nach der Fundstelle anzuzeigen.

`-A`*n*	(**a**fter)	Gibt die folgenden *n* Zeilen nach der Fundstelle aus.
`-B`*n*	(**b**efore)	Gibt die vorausgehenden *n* Zeilen vor der Fundstelle aus.
`-C`*n*	(**c**enter oder **c**ontext)	Gibt *n* Zeilen vor und nach der Fundstelle aus.
`-m`*n*	(**m**ax count)	Die Durchsuchung einer Datei endet bei *n* Treffern (Mit `-m1` kann die Suche in großen und bei vielen Dateien beschleunigt werden)

Alle `grep`–Programme setzen den Exitstatus auf:

0 wenn das Muster gefunden wurde
1 wenn kein Muster gefunden wurde
2 wenn ein Fehler passierte

Beispiele:

- Alle Dateien auflisten die von allen Benutzern gelesen und beschrieben werden können.
```
$ ls -l | grep '^.......rw'
```
- Alle Zeilen in den Dateien *.txt finden, in denen bestimmte Worte benutzt werden. Die Datei *words* muss alle gesuchten Worte enthalten.
```
$ fgrep -fwords *.txt
```
- „Lines of code“ ermitteln, ohne Leerzeilen zu berücksichtigen.
```
$ grep -h -v '^[ <TAB>]*$' *.c | wc -l
```
- In der Datei `/etc/passwd` soll nachgeschaut werden, ob der Benutzer "hoz" dem System bekannt ist.
```
$ grep -i "^hoz:" /etc/passwd
```
- Alle Dämonprozesse des Systems anzeigen.
```
$ ps -ef | grep ' ? '
```
- Alle Prozesse anzeigen, die auf "sh" enden.
```
$ ps -e | grep 'sh$'
```
- Beispiele für die Context Optionen:
```
$ seq 10 | grep -A 1 5
5
6
$ seq 10 | grep -B 1 5
4
5
$ seq 10 | grep -C 1 5
4
5
6
```

6.5.1 Aufgaben

1. Geben sie den *ssh*(1) Schlüssel des Unix Login Servers auf dem Bildschirm aus.
2. Ermitteln sie die Anzahl der ECDSA, sowie der RSA Schlüssel aller *ssh*(1) Server die sie bereits kontaktiert haben.

6.5.2 Nicht standard grep Kommandos

Neben der Unix/GNU grep Familie existieren eine ganze Reihe weiterer grep ähnlicher Kommandos. An dieser Stelle soll beispielhaft lediglich auf ein bestimmtes Kommando eingegangen werden, da es sich hierbei um eine in vielerlei Hinsicht sinnvolle Erweiterung handelt. Das Kommando ist nicht in C sondern in der Programmiersprache Rust geschrieben. Der Name des Projektes ist *ripgrep*, allerdings wird das Kommando selbst über *rg*(1). aufgerufen.
Syntax:

rg [*optionen*] *suchmuster* [*path* ...]

Der wesentliche Unterschied ist, dass dieses Kommando, im Gegensatz zu den originären Unix *grep*(1) Kommandos, einen gesamten Dateibaum nach bestimmten Mustern durchsucht. Damit erspart man sich den etwas kruden und fehleranfälligen Aufruf mit *find*(1). Statt

```
$ find . -exec grep <muster> /dev/null  ;
```

Erreicht man ähnliches mit

```
$ rg <muster> .
```

Die grundlegenden Optionen sind an die standard grep-Kommandos angelehnt. Die regulären Ausdrücke entsprechen weitestgehend denen von *egrep*(1).
Die meisten Neuerungen beziehen sich auf die Fähigkeit einen ganzen Dateibaum zu durchsuchen, und damit löst *rg*(1) die doch recht komplizierte Kombination von *grep*(1) mit *find*(1) ab. Daneben ist die Ausgabe farbig gestaltet und passt sich an den Ausgabekanal (stdout oder Datei bzw. Pipeline) an.

Einige der neuen Optionen von *rg*(1):

`-t` *`filetype`*
`--type` *`filetype`*
: Sucht nur in Dateien deren Dateimuster dem angegebenem Dateityp entspricht. Der Dateityp **c** sucht z.B. ausschließlich in `.c` und `.h` Dateien.[18]
Damit kann man sehr einfach den komplizierten Aufruf mit grep und find ersetzen:

```
$ find . \( -name "*.c" -o -name "*.h" \) \
     -exec grep <muster> /dev/null \{\} \;
$ rg -t c <muster>
```

`-T` *`ftype`*
`--type-not` *`ftype`*
: Durchsucht alle Dateien die nicht dem angegebenen Typ angehören

`--type-list`
: Gibt eine Liste der aktuell unterstützten Dateitypen mit den zugeordneten Dateinamensmuster aus. Um zu sehen welche Dateimuster einem Typ zugeordnet sind kann man die Ausgabe mit *rg*(1) selbst filtern:

```
$ rg --type-list | rg "^c:"
c: *.[chH], *.[chH].in, *.cats
```

`-i`
`--ignore-case`
: Ignoriert Groß- und Kleinschreibung (wie bei grep auch)

18. Für ein genauere Information welche Dateimuster zu einem Dateityp zugeordnet sind siehe `--type-list`

`-S`
`--smart-case` Ignoriert Groß- und Kleinschreibung wenn das Muster komplett klein geschrieben ist, sonst wird exakt nach dem Muster gesucht

`-n`
`--line-number` Vor jeder Zeile steht, durch Doppelpunkt getrennt, die Zeilennummer der Fundstelle

`-N`
`--no-line-number` Unterdrückt die Ausgabe der Zeilennummer der Fundstelle

`--column` Vor jeder Zeile steht, durch Doppelpunkt getrennt, die Spaltennummer der Fundstelle. Dies stellt allerdings leider die Bytenummer dar, nicht die Position des (Unicode) Zeichens in der Zeile.

```
$ echo "aöc" | rg -N --column "a"
1:aöc
$ echo "aöc" | rg -N --column "ö"
2:aöc
$ echo "aöc" | rg -N --column "c"
4:aöc
```

`--max-depth` Gibt die maximal zu durchsuchende Tiefe des Dateibaums an.

`--vimgrep` Gibt vor jeder Zeile mit einem Treffer den Dateinamen, die Nummer der Zeile in der Datei, sowie die Spaltennummer aus. Zeilen mit mehr als einem Treffer werden mehrfach ausgegeben.

6.6 Reguläre Ausdrücke

Als reguläre Ausdrücke (regular expressions) bezeichnet man Muster, die zur Suche in Texten verwendet werden. Diese Muster können bei verschiedenen Unix−Kommandos angegeben werden:

- Bei den Kommandos `vi`, `ed`, `pg`, `more` und `less` zur Suche nach Textstellen (**/***muster***/** bzw. **?***muster***?**).
- Bei `sed`, `ed` und `vi` als Muster bei dem Substitute Kommando (**s/***muster***/***ersetzung***/**).
- Bei den `grep`-Kommandos zum Extrahieren von Zeilen aus Dateien.
- Bei den Kommandointerpretern (`sh`, `ksh`, `csh`) zur Angabe von Dateimustern.
- Bei `lex` zur Konstruktion von sog. Eingabescannern (lexikalische Analyse). Wird für die Programmierung von Compilern verwendet.

6.6.1 Übersicht über reguläre Ausdrücke bei den verschiedenen UNIX−Kommandos

Bedeutung	*Shell*	*ed/sed*	*ex/vi*	*grep*	*egrep*	*awk*	*lex*
bel. Zeichen	`?`	`.`	`.`	`.`	`.`	`.`	`.`
bel. Zeichenkette (auch die leere)	`*`	`.*`	`.*`	`.*`	`.*`	`.*`	`.*`
bel. Wiederholung des vorangegangenen Zeichens (auch keine)		`*`	`*`	`*`	`*`	`*`	
Wiederholung (>=1)		`\{1\}`			`+`	`+`	`+`
Wiederholung (0|1)		`\{0,1\}`	`\{0,1\}`				
Wiederholung (=m)		`\{m\}`	`\{m\}`				
Wiederholung (>=m)		`\{m,\}`	`\{m,\}`				
Wiederholung (>=m und <=n)		`\{m,n\}`	`\{m,n\}`				
Zeichenklasse	`[-]`	`[-]`	`[-]`	`[-]`	`[-]`	`[-]`	`[-]`
Negation d. Klasse	`[!]`	`[^]`	`[^]`	`[^]`	`[^]`	`[^]`	`[^]`
Zeilenanfang		`^`	`^`	`^`	`^`	`^`	`^`
Zeilenende		`$`	`$`	`$`	`$`	`$`	`$`
Wortanfang			`\<`				
Wortende			`\>`				
Alternative					`|`	`|`	`|`
Gruppierung					`()`	`()`	`()`
Fluchtsymbol	`\`	`\`	`\`	`\`	`\`	`\`	`\`

7. Kommunikation & Netzdienste

7.1 Benutzerkommunikation

7.1.1 Benutzerkommunikation mit Hilfe der Ausgabeumlenkung

Da ein Terminal als Gerätedatei behandelt wird (genau wie der Drucker), kann man die Ausgabeumlenkung dazu benutzen, Nachrichten auf ein fremdes Terminal zu senden.

```
$  cat hinweis > /dev/pts/1
→ Schreibt den Inhalt der Datei "hinweis" auf das Terminal "pts/1".

$  cat > /dev/console
Meldung an den Systemverwalter.
^D
→ Schreibt den nachfolgenden Text auf die Systemverwalterconsole.
```

Den Namen des Terminals erfährt man über das Kommando `tty`.

Damit nicht einfach ein Benutzer bei seinem Nachbarn den Bildschirm vollschreiben kann, ist der Zugriff auf die Terminalgeräte eingeschränkt. Die Ausgabeumlenkung funktioniert nur dann, wenn man die entsprechende Berechtigung (Schreibrecht) dazu hat.

```
$ ls -l /dev/pts
crw------- 1 hoz    tty 136, 0 Aug 28 15:12 0
crw--w---- 1 hugo   tty 136, 1 Aug 28 15:04 1
crw------- 1 gustav tty 136, 2 Aug 28 15:12 2
```

Im Beispiel wurde der Gruppe `tty` Schreibrecht auf das Gerät `pts/1` gegeben. Ob ein Benutzer das Versenden von Nachrichten erlaubt hat kann durch das Kommando *who* mit der Option `-T` ermittelt werden.

```
$ who -T
hoz          ? :0           Aug 28 14:44 (console)
hoz          - pts/0        Aug 28 14:44
hugo         + pts/1        Aug 28 14:44
gustav       - pts/2        Aug 28 14:44
```

7.1.2 Nachrichtenankunft steuern

Will ein Benutzer ungestört an seinem Terminal arbeiten, hat er die Möglichkeit, die Nachrichtenankunft zu sperren.

mesg (Control display of messages)

Syntax:

mesg [*y*|*n*]

Wird `mesg` ohne Parameter aufgerufen, gibt es an, ob Nachrichten empfangen werden können oder nicht.

Beispiel:

```
$  mesg
is y

$  mesg n

$  mesg
is n
```

Die Einstellung des Kommandos `mesg` gilt nur für die aktuelle Sitzung.

7.1.3 Versenden von Nachrichten

Der Befehl `write` arbeitet prinzipiell wie eine Ausgabeumlenkung.

Beim Versenden von Nachrichten muss allerdings nicht mehr der Terminalname ermittelt werden.
Der Kommunikationsaufbau erfolgt über den Benutzernamen.

Arbeitet ein Benutzer an mehreren Terminals, kann zusätzlich der Terminalname angegeben werden.

write (write to user)

Syntax:

write *benutzer* [*terminal*]

Beispiel:

```
$  write hoz
Alles, was jetzt getippt wird,
erscheint auf dem Terminal
des Benutzers hoz.
^D
```

Bei dem Benutzer "hoz" erscheint auf dem Terminal der Text:

```
Message from Benutzername@Hostname on terminal at uhrzeit
Alles, was jetzt getippt wird,
erscheint auf dem Terminal
des Benutzers hoz.
(end of message)
```

Mit `write` können nur Nachrichten an eingeloggte Benutzer gesendet werden.

7.1.4 Nachrichten an alle eingeloggten Benutzer senden

wall (write to all users)

Syntax:

wall

7.2 Elektronische Post (mail)

Unterschied zu `write`:

- Nachrichten kommen nicht sofort auf den Bildschirm des Empfängers, sondern werden in einem Briefkasten gespeichert.
- Der Empfänger erhält lediglich eine Nachricht, dass Post für ihn eingetroffen ist.
- Der Empfänger kann selber entscheiden, ob und wann er die eingegangene Post bearbeiten möchte.

Die Benachrichtigung über eingegangene Post lautet:

```
"you have mail"
```


7.2.1 Versenden von Briefen

mail (send mail to user)

Syntax:

mail *benutzername* [...]

oder

mailx [**–s** *betreff*] *benutzername* [...]

Der zu sendende Brief wird von der Standardeingabe gelesen.

Der Brief wird beendet durch `<^D>` am Zeilenanfang (Bei manchen Systemen auch durch eine Zeile, die am Anfang einen Punkt enthält).

Ein vorgefertigter Brief kann mit Hilfe der Eingabeumlenkung versendet werden.

```
$  mail adresse < brief
```

7.2.2 Bearbeiten der Briefe

Alle 10 Minuten wird in dem eigenen Briefkasten nachgeschaut, ob Post angekommen ist.

Diese Zeitspanne kann verändert werden über:

```
$  MAILCHECK=neue_Zeit_in_Sekunden
$  export MAILCHECK
```

mail (look at your mailbox)

Syntax:

mail [**–hr**]

Optionen:

`-h`	(header)	Es werden nur die Briefköpfe angezeigt.
`-r`	(reverse)	Die Briefe werden in der Reihenfolge ihres Eintreffens angezeigt.

Beispiel:

```
$  mail
From:  absender_name  datum
Hier folgt der Brief
? (Hier können jetzt Befehle eingegeben werden)
```

Die Möglichkeiten des Kommandos `mail` sind versionsabhängig.
Hier ist lediglich eine simple Version des Kommandos dargestellt. Die meisten Mailprogramme geben eine Hilfeseite aus, wenn man ein Fragezeichen eingibt. Für weiterführende Angaben sei auf die Manualseite verwiesen.

7.3 Netzdienste

7.3.1 Remote Terminaldienste

Da Unix ein Multitasking- und Multiuser Betriebsystem ist, war es von Anfang an möglich sich über Datennetzverbindungen von anderen Rechnern aus auf Unix Maschinen anzumelden und dort zu arbeiten. Alle Kommandos werden dann auf dem entfernet Rechner ausgeführt, während die Ein- und Ausgaben auf dem originären Rechner stattfinden.
Natürlich kann ein Unix Rechner auch als Client für eine solche „Remote Terminal Session“ verwendet werden.

telnet (Terminal Emulation over Network)

Syntax:

telnet [**–l** *user*] *host* [*port*]

Achtung: Das Kommando sollte heute nicht mehr, oder nur noch in kontrollierten Umgebungen verwendet werden da sämtliche Ein- und Augaben, d.h. auch das Passwort, im Klartext übertragen werden!
Für einfaches Netzwerk debugging ist es aber immer noch gut geeignet:

```
$ telnet www.hznet.de 80
Trying 213.239.204.36...
Connected to www.hznet.de.
Escape character is '^]'.
GET /
...
Connection closed by foreign host.
```

rsh (remote shell)

Syntax:

rsh *host* [**–l** *user*] [*command*]

Im Unterschied zu *telnet*(1) erlaubt das Kommando *rsh* ein Anmelden auf dem entfernten Rechner ohne das ein Passwort angegeben werden muss.
Achtung: Das Kommando sollte heute nicht mehr verwendet werden, da der Mechanismus zur Authentifizierung extrem leicht überwunden werden kann.

ssh (secure shell)

Syntax:

ssh [**–l** *user*] *host* [*command*]

Ein Terminal Emulationsprogramm mit dem eine sichere Authentifizierung und verschlüsselte Verbindung mit dem entfernten System ermöglicht. Dies ist heute das Standardprogramm für Remote Sessions. Bei entsprechender Konfiguration ist auch eine sichere, Passwort freie Authentifizierung möglich. Bei der Angabe eines Kommandos wird dieses auf dem entfernten Rechner ausgeführt, und die die Ausgabe des Kommandos lokal angzeigt:

```
hoz@xt5:~> ssh xt4.hznet.de "date; hostname; who"
Thu Jun 19 23:59:23 CEST 2008
xt4.hznet.de
hoz      tty1         Jun 19 23:33
```

7.3.2 Dateiübertragung

Das klassische Kommando zur Übertragung von Dateien ist *ftp*(1).

ftp (file transfer program)

Syntax:

ftp *host*

FTP ist ein interaktives Programm. Es hat die gleichen Probleme wie telnet, da auch bei diesem Programm die Übertragung des Passwortes im Klartext erfolgt. Daher sollte das Programm nur noch für sog. *annonymous* FTP verwendet werden.

scp (secure copy)

Syntax:

scp *[host:]src [host:]dest*

Ermöglicht das Kopieren von Dateien auf oder von entfernten Rechnern. Man benötigt auf dem Rechner eine Benutzerkennung. Das Kommando gehört zu dem Kommando *ssh*(1) und nutzt die gleiche Infrastruktur. Wer sich per *ssh* auf einem entfernten System anmelden kann, der kann auch mit *scp* Dateien dorthin kopieren. Entspricht diese nicht der lokalen Benutzerkennung muss vor der Angabe des Rechners noch der Benutzer mit einem @-Symbol angegeben werden.

```
$ scp "hoz@xt4.hznet.de:*.txt" .
```

Meldet sich unter dem Namen hoz auf dem Rechner xt4.hznet.de an und kopiert aus dem Heimatverzeichnis alle Dateien mit der Kennung .txt in das aktuelle Verzeichnis.

wget (non interactive network downloader)

Syntax:

wget [*optionen*] *url*

Das Kommando kann mit unterschiedlichen Netzprotokollen umgehen, und so (öffentlich zugängliche) Dateien von entfernten Rechner kopieren. Die unterstützten Protokolle sind unter anderem *FTP* und *HTTP*. Es erlaubt das rekursive Laden von verlinkten Webseiten bis zu einer voreinstellbaren Tiefe. Man kann damit sehr einfach eine komplette Webseite lokal spiegeln. Typisches Anwendungsgebiet ist jedoch das Laden von Programmpaketen von remote FTP Servern:

```
$ wget ftp://ftp.gnu.org/pub/gnu/packages-name
```

curl (transfer a URL)

Syntax:

curl [*options*] *url*

Das Kommando ist eine Alternative zu *wget*(1) falls dieses auf dem Unix System nicht installiert ist (z.B. unter MacOSX).
Möchte man damit ein Programmpaket von einem entfernten Webserver laden, sollte mit der Option `-O` eine Ausgabedatei spezifiziert, oder mit einer Ausgabeumlenkung die Standardausgabe in eine Datei gelenkt werden.

Die Kommandos *curl*(1) und *wget*(1) haben definitv zu viele Optionen.

lynx (non graphical web browser)

Syntax:

lynx [*optionen*] *url*

Ein text basierter Webbrowser. Heutzutage sehen die meisten Webseiten, wenn nur der reine Text dargestellt wird, zumindest „merkwürdig“ aus. Viele können jedoch gar nicht dargestellt werden (z.B. Flash basierte). Trotzdem ist das Kommando zum Testen und zur Anzeige von einfachen Webseiten gut zu gebrauchen.

w3m (non graphical web browser)

Syntax:

w3m –help
w3m –show-option
w3m [**–4|6**] [**–dump**] [**–dump_head**] [**–dump_source**] [*optionen*] *url*

IPv6 fähiger, text basierter Webbrowser und Pager. Die *dump*-Optionen können genutzt werden um die Webseite als Text resp. den Sourcecode auf die Standardausgabe zu schreiben.

8. Unix – Kommandoübersicht

Hier wird eine Auswahl von Unix Kommandos aus völlig unterschiedlichen thematischen Bereichen vorgestellt.

age (actually good encryption)

Syntax:

age **–e** **–r** *recipientkey* | **–R** *rkeyfile* [**–rR** *recipient*] [*datei*]
oder
age **–d** **–i** *keyfile*|– [*datei*]

age(1) dient zur Verschlüsselung einer Datei für ein oder mehrere Empfänger. Es ist kein Standard Unix Kommando!
Das Programm nutzt zur Verschlüsselung einen zufälligen symmetrischen Schlüssel, der mit den öffentlichen Schlüsseln der Empfänger verschlüsselt wird. Damit kann die Datei von mehreren Empfängern entschlüsselt werden, ohne das geheime Schlüssel ausgetauscht werden müssen. Der öffentliche Teil des Empfängerschlüssels kann von diesem über Klartextkanäle übertragen werden.
Der Empfänger benötigt zugriff auf den (seinen) privaten Schlüssel. In der Regel hat dieser ihn in einer Datei bei sich hinterlegt, und die Zugriffsrechte darauf eingeschränkt. Optional kann die private Shclüsseldatei mit einem symmetrischen Schlüssel verschlüsselt sein.

Beispiel:

```
# Auf Seite des "Senders"
$ date | age -e -R empf-keyfile -o test.age
$ ls -l test.age
-rw-rw-r-- 1 hoz hoz 230 Sep 26 11:17 test.age
# Übermittlung der Datei zum Empfänger über öffentliche Kanäle

# Auf Seite des "Empfängers"
$ age -d -i agekeyfile test.age
So 26. Sep 11:17:17 CEST 2021

# Mit verschlüsselter Schlüsseldatei
$ age -d -i agekeyfile test.age
Enter passphrase for identity file:
So 26. Sep 11:17:17 CEST 2021
```

banner

Syntax:

banner *text* [...]

Gibt *text* in Großschrift auf der Standardausgabe aus.

Es gibt zwei unterschiedliche Ausführungen des *banner* Kommandos: Die SysV Version gibt den Text horizontal (auch für Bildschirm geeignet) aus, wohingegen die BSD Varinate den Text vertikal ausgibt. Letzteres eignet sich für eine Ausgabe auf (Endlos)Druckern. Die SysV Variante begrenzt den Text auf eine Länge von maximal 10 Zeichen, und gibt weitere angegebene Textparameter untereinander auf der Standardausgabe aus.

Unter Linuy existiert kein *banner*(1) Kommando. Als wesentlich vielseitigerer Ersatz kann *figlet*(1) herangezogen werden.

bc (basic calculator)

Syntax:

bc [**–l**]

Ein Programm zur Berechnung von mathematischen Ausdrücken. Die Ausdrücke werden von der Standardeingabe gelesen. Die Syntax ist mehr oder weniger so wie man es erwarten würde. Das Kommando `quit` oder `<^D>` beendet das Programm.

```
$  bc
 2 + 4 * 8 - 3
31
quit
$ echo "2 + 4 * 8 - 3" | bc
31
```

Achtung:
`bc` kann mit sehr vielen Stellen nach dem Komma rechnen, zeigt aber im Normalfall keine Nachkommastellen an. Legt man Wert auf Nachkommastellen, sollte die Option `-l` angegeben werden, oder die Variable `scale` auf die Anzahl der Nachommastellen gesetzt werden.

Sowohl die Ein- als auch die Ausgabe von `bc` kann in beliebigen Zahlensystemen erfolgen. Das Eingabe Basissytem wird durch `ibase=` gesetzt. Entsprechend das Ausgabebasisystem mit `obase=`.

In der Originalversion unterstützt *bc*(1) lediglich Variablen die aus einem Buchstaben bestehen. In der GNU Version von *bc*(1) können Variablennamen auch länger sein.
Die GNU Version ermöglicht das Editieren der Eingabezeile mit Hilfe der GNU Readline Bibliothek. Diese entspricht der *bash*(1), sodass hier die Syntax identisch ist.

cal (print calendar)

Syntax:

cal [*[monat] jahr*]

Erzeugt einen Kalender für das angegebene Jahr (Jahresangabe mit Jahrhundert).
Werden zwei Parameter angegeben, bestimmt der erste Parameter den Monat des Jahres.

ncal (print calendar)

Syntax:

ncal [**–3MSew**] [**–m** *monat*] [**–y**] [*[monat] jahr*]

Das Kommando *ncal*(1) ist eine Weiterentwicklung des *cal*(1) Kommandos. Es versteht mehr Optionen (u.a. eine Berechnug des Osterfestes) und kann den aktuellen Tag in der Ausgabe hervorheben.

- `-w` Zeigt die Wochennummer mit an
- `-S` Die Woche beginnt Sonntags
- `-M` Die Woche beginnt Montags
- `-M` Das Datum des Osterfestes im aktuellen oder angegebenen Jahr.
- `-3` Gibt drei Monate nebeneinander aus. Default: Den aktuellen bzw. den spezifizierten Monat, bzw. ein ganzes Jahr falls eine Jahresangabe vorgenommen wurde.

`-m monat`
: Gibt den angegebenen Monat aus.

`-y` Erzeugt eine Jahresausgabe (12 Monate).

Beispiel:

```
$ ncal -3 -w
    Juli 2015          August 2015        September 2015
Mo     6 13 20 27         3 10 17 24 31      7 14 21 28
Di     7 14 21 28         4 11 18 25      1  8 15 22 29
Mi  1  8 15 22 29         5 12 19 26      2  9 16 23 30
Do  2  9 16 23 30         6 13 20 27      3 10 17 24
Fr  3 10 17 24 31         7 14 21 28      4 11 18 25
Sa  4 11 18 25         1  8 15 22 29      5 12 19 26
So  5 12 19 26         2  9 16 23 30      6 13 20 27
   27 28 29 30 31     31 32 33 34 35 36 36 37 38 39 40
```

date (set/get date and time)

Syntax:

date *MMTThhmmJJ*

oder

date [+*format*]

Mit der ersten Form des Kommandos kann das Datum und die Uhrzeit neu gesetzt werden (darf nur der Systemverwalter).

Die zweite Form dient zum Anzeigen des aktuellen Datums bzw. der Uhrzeit. `date` verwendet dafür ein Standardformat.

Beispiel:

```
$  date
Tue Aug 18 13:20:36 CEST 2009
```

Das Ausgabeformat kann durch den Formatparameter frei bestimmt werden. Nach dem Pluszeichen kann beliebiger Text angegeben werden oder besondere Zeichenfolgen, die von `date` durch Datumskomponenten ersetzt werden.

Der Formatparameter sollte in Anführungszeichen gesetzt werden.

Format-kennzeichen	*Bedeutung*
`%A`	Wochentagsname (Sontag, Montag usw.)
`%a`	abgekürzter Wochentagsname (Son, Mon usw.)
`%B`	Monatsname (Januar, Februar usw.)
`%b`	abgekürzter Monatsname (Jan, Feb usw.)
`%d`	Tagesangabe (1–31)
`%m`	Monatsangabe (1–12)
`%y`	Jahresangabe (00–99)
`%H`	Stundenangabe (00–23)
`%M`	Minutenangabe (00–59)
`%S`	Sekundenangabe (00–59)
`%T`	Zeitangabe (hh:mm:ss)

Beispiel:

```
$  date +"Today is %A, %B the %d. %Y (%H:%M)"
Today is Tuesday, August the 18. 2009 (13:23)
```

Das Kommando *date*(1) wertet die Umgebungsvariable `TZ` zur Spezifikation der angezeigten Zeitzone aus. Mit der Option `-u` wird immer die koordinierte Weltzeit UTC[19] ausgegeben.

Beispiel:

```
$ date -u
Fr 7. Jul 09:24:01 UTC 2017
$ date
Fr 7. Jul 11:24:21 CEST 2017
$ TZ='Asia/Shanghai' date
Fr 7. Jul 17:24:28 CST 2017
$ TZ='America/New_York' date
Fr 7. Jul 05:24:32 EDT 2017
```

basename

Syntax:

basename *pathname* [*extension*]

Das Kommando entfernt bei dem angegebenen Pfadnamen den Verzeichnisnamen.

Ist zusätzlich eine Extension angegeben, wird diese ebenfalls entfernt.

Beispiel:

```
$  echo $MAIL
/usr/spool/mail/hoz
$  basename $MAIL
hoz
$  basename $HOME/src/c/bsp.c .c
bsp
```

dirname

Syntax:

dirname *pathname*

Das Kommando entfernt bei dem angegebenen Pfadnamen den Dateinamen.

```
$  echo $MAIL
/usr/spool/mail/hoz
$  dirname $MAIL
/usr/spool/mail
$ dirname $HOME/src/c/bsp.c
/home/hoz/src/c
```

echo

Syntax:

echo [*text ...*]

`echo` gibt den angegebenen Text mit nachfolgendem Zeilenvorschub auf dem Bildschirm aus.

Der Text sollte in Anführungszeichen geschrieben werden, um Zeichen mit einer Sonderbedeutung zu schützen.

19. UTC ⇒ Unviversal Time Coordinated; CEST ⇒ Central European Summer Time; CST ⇒ China Standard Time; EDT ⇒ Estern Daylight Time; Siehe auch https://de.wikipedia.org/wiki/Zeitzone

id (print user and group identification)

Syntax:

id

Beispiel:

```
$  id
uid=201(hoz) gid=200(kaho)
```

file (guess file type)

Syntax:

file *datei* [...]

Das Kommando versucht zu erraten, um welche Art von Datei es sich handelt. Einige mögliche Dateiarten:

Ausgabe von file	*Bedeutung*
`ascii text`	Text Datei
`c program text`	C–Quelldatei
`commands text`	Batchdatei
`data`	Binärdatei
`directory`	Inhaltsverzeichnis
`empty file`	Leere Datei
`executable`	Ausführbare Datei (z.B. übersetztes C–Programm)

man (print manual page)

Syntax:

man [*kapitel*] *begriff*

Zeigt die Manualseite zu dem angegebenen Begriff auf dem Bildschirm an.

Die Manualseiten sind in Kapiteln zusammengefasst (z.B. stehen alle Kommandos in Kapitel `1`).

Kapitel	*Kapitelname*	*Beschreibung*
1	Commands	Kommandobeschreibung
8	Administration	Kommandos für den Systemverwalter
5	File formats	Beschreibung von Dateien (z.B. passwd)
4	Hardware	Beschreibung von Geräten (z.B. Floppy)
7	Miscellaneous	Sonstiges (z.B. Makropakete)
2	System calls	Kernel C–Funktionen
3	Library calls	C–Funktionen

Einige Begriffe stehen in mehreren Kapiteln. So gibt es sowohl ein Kommando `passwd` als auch eine Datei mit dem Namen `passwd`.

Bei Aufruf des `man` Kommandos muss in diesme Fall als erster Parameter das Kapitel angegeben werden aus dem die Beschreibung angezeigt werden soll.

Beispiel:

```
$ man 5 passwd
```

Wird *man*(1) mit der Option `-k` aufgerufen, wird nach dem angegeben Begriff in allen Manualseiten gesucht:

```
$  man -k grep
egrep (1)     - print lines matching a pattern
fgrep (1)     - print lines matching a pattern
grep (1)      - print lines matching a pattern
grep (1p)     - search a file for a pattern
mgrep (1)     - search mimencoded files for a regular expression
msggrep (1)   - pattern matching on message catalog
pcregrep (1) - a grep with Perl-compatible regular expressions.
pgrep (1)     - look up or signal processes based on name and other attributes
zgrep (1)     - search possibly compressed files for a regular expression
zipgrep (1)   - search files in a ZIP archive for lines matching a pattern
```

nice

Syntax:

nice *kommando*

Startet *kommando* mit einer niedrigereren Priorität. Ist das nicht „nett“ auf einem Multiuser System? Bei *nice*(1) handelt es sich um ein eher selten genutztes Kommando.

nohup (no hangup)

Syntax:

nohup *kommando*

Startet *kommando* so, dass man sich ausloggen kann, ohne dass dabei das Kommando abgebrochen wird. Um sich ausloggen zu können, sollte man die gesamte Kommandofolge im Hintergrund ausführen. Bei *nohup*(1) handelt es sich um ein eher selten genutztes Kommando.

od (octal dump)

Syntax:

od [**−c**] [**−A** *Radix*] [**−t** *Format*] [*datei*]

Zeigt die angegebene Datei byteweise oktal auf der Standardausgabe an. Die Option `-c` sorgt für eine Darstellung, bei der nichtsichtbare Zeichen als 3-stellige Oktalzahl dargestellt werden.

Es gibt auch hier eine Vielzahl von Optionen um auch zwei- oder vierbyte Werte in unterschiedlichen Zahlensystemen (`-t`) ausgeben zu können. Das *Format* bestimmt die Art der Ausgabe des Inhaltes der Datei. Mit `dox` wird das Zahlensystem gewählt und über eine angehängte Ziffer die Anzahl der darzustellenden Bytes.
Das Zahlensystem für die Angabe des Offset wird über die Option `-A` bestimmt. *Radix* kann dabei eines der Buchstaben `doxn` für dezimale, oktale, hexadezimale oder keine (none) Ausgabe des Offset sein.

Eine klassische Hexausgabe ergibt sich über

```
$ od -Ad -tx1z <file>
```

seq (print sequence)

Syntax:

seq [**−s** *separator*] [**−w**] [*first* [*increment*]] *last*

Druckt eine Zahlenfolge beginnend bei *first* (default ist eins) bis *last*. Die Zahlen werden jeweils um *increment* (default ist eins) erhöht. Ist der Startwert größer als der Endwert, wird als *increment* -1 benutzt.

Die Option `-w` erzeugt gleich lange Zahlen indem diese mit führenden Nullen ausgegeben werden. durch die Option `-s` wird das Trennzeichen zwischen den Zeichen festgelegt. Standard ist ein Zeilenvorschub. Es kann auch ein leeres Trennzeichen angegeben werden.

Beispiel:

```
$ seq -s " " 0 5 99
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
```

Die Generierung der Zahlenfolge ist nicht auf ganze Zahlen beschränkt. Man beachte hier die inkonsistente Angabe des Inkrements mit Dezimalpunkt bei gleichzeitiger Ausgabe der Werte als (im deutschen üblichen) Kommawerte. Zumindest letzteres lässt sich durch das Setzen der LC_NUMERIC Variable anpassen.

```
$ seq -s " " 0 0.5 5
0,0 0,5 1,0 1,5 2,0 2,5 3,0 3,5 4,0 4,5 5,0
$ LC_NUMERIC=C seq -s " " 0 0.5 5
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

sleep (suspend execution)

Syntax:

sleep *sekunden*

Dieses Kommando stellt einen Wecker auf die angegebene Anzahl von Sekunden und legt sich „schlafen“. Dabei wird das System nicht belastet. Nach Ablauf des Weckers wird dasa Kommando „geweckt“ und der Prozess terminiert.

Auch wenn es nicht so aussieht, kann dieses Kommando sehr nützlich sein.

stty (show/set terminal characteristics)

Syntax:

stty [**–a**]

oder

stty sane

Das Kommando `stty` zeigt die Einstellung des Terminaltreibers auf dem Bildschirm an. Interessant sind dabei höchstens die Einstellung der Unterbrechungs– (intr), Lösch– (erase) und Lösche Zeile (kill) Taste.

Die zweite Form des Kommandos kann verwendet werden, um ein Terminal, dass sich „merkwürdig“ verhält (z.B. keine Darstellung von getippten Zeichen), wieder in einen definierten Zustand zu versetzen.

tee

Syntax:

tee *datei* [...]

Das Kommando bildet in einer Pipeline eine Art T-Stück (Abzweigung).

Das Kommando ließt Daten von der Standardeingabe und schreibt sie auf die Standardausgabe. Zusätzlich wird eine Kopie der Eingabe in *datei* abgelegt.

Beispiel:

```
$  ls /bin /usr/bin | tee unsort | sort
```

time

Syntax:

time *kommando*

Startet das *kommando* und gibt den Zeitbedarf auf der Standardfehlerausgabe aus.

Beispiel:

```
$  time cc bsp.c
```

tty (print tty name)

Syntax:

tty

Gibt den Pfadnamen des Gerätetreibers aus, mit dem das Terminal verbunden ist.

uname (unix name)

Syntax:

uname [**–rv**]

Gibt den Namen des UNIX–Rechners auf der Standardausgabe aus.

Die Option `-r` gibt die Release des UNIX–Systems und die Option `-v` die Version aus.

wc (word count)

Syntax:

wc [**–clw**] [*file* ...]

Ermittelt die Anzahl der Zeilen, Zeichen und Wörter von Dateien oder der Standardeingabe. Ein Wort ist dabei eine durch *white space* getrennte Folge von beliebigen Zeichen.

Die Option `-c` beschränkt sich auf das ermitteln der Zeichen, `-l` auf die Zeilen und `-w` auf die Wörter der Dateien.

Beispiele:

```
$  wc -l < /etc/passwd
     78

$  cd src/c
$  wc -l *.c
      7 t1.c
      9 t2.c
      7 t3.c
     23 total
```

who (who is online)

Syntax:

who [**–abHqTu**]

oder

who am i

`who` gibt eine Liste aller momentan eingeloggten Benutzer aus.

`-a`	(all)	Alle Informationen ausgeben, die zur Verfügung stehen
`-b`	(Boot)	Zeigt den Zeitstempel des letzten Systemboot an
`-H`	(Header)	Schreibt eine Überschrift über die Ausgabespalten
`-q`	(quick)	Kurze Liste aller eingeloggten Benutzer
`-T`	(Terminal)	Gibt über ein '+' oder '-' Zeichen an ob der Nutzer Nachrichten empfangen kann
`-u`	(useful)	Zusatzinformationen über die Benutern anzeigen

9. Unix – Einzeiler

In diesem Kapitel werden sinnvolle (und manchmal auch weniger sinnvolle) Kommandofolgen dargestellt. Ziel ist es, dass Zusammenwirken unterschiedlicher Kommandos zu zeigen, um damit ganz neue Funktionalitäten zu erreichen.
Sollte sich die Kommandofolge als hilfreich erweisen, kann diese in Form eines Shell Skriptes in einer (Batch)Datei abgelegt und bei Bedarf aufgerufen werden.

Eine lange Zeichenfolge umbrechen

Die Problemstellung ergab sich aus einer BASE64 codierten Zertifikatsdatei, deren sehr lange Zeichenfolge anschaulich umgebrochen und dargestellt werden sollte.

Zur Simulation der Eingabedaten wird hier eine lange Zeichenfolge generiert und in einer Variablen gespeichert:

```
$ longstr=`seq -s "" 180`
```

Mit *echo* kann dieser Teststring angezeigt werden. Zur Aufbereitung wird *sed*(1) genutzt um nach jeweils 8 Zeichen ein Leerzeichen einzufügen. Das Kommando *fold*(1) trennt dann die (sehr lange) Zeile in mehrere Zeilen auf, der Parameter `-w` begrenzt die Anzahl der Zeichen pro Zeile:

```
$ echo $longstr | sed 's/\(........\)/\1 /g' | fold -s -w 50
12345678 91011121 31415161 71819202 12223242
52627282 93031323 33435363 73839404 14243444
54647484 95051525 35455565 75859606 16263646
56667686 97071727 37475767 77879808 18283848
58687888 99091929 39495969 79899100 10110210
31041051 06107108 10911011 11121131 14115116
11711811 91201211 22123124 12512612 71281291
30131132 13313413 51361371 38139140 14114214
31441451 46147148 14915015 11521531 54155156
15715815 91601611 62163164 16516616 71681691
70171172 17317417 51761771 78179180
```

Tabulator, oder andere Zeichen in eine Folge von Leerzeichen ersetzen

Als Beispiel wird hier die Datei `/etc/group` benutzt. In dieser sind einzelne Felder durch Doppelpunkt getrennt. Dieser Doppelpunkt kann durch eine entsprechende Folge von Leerzeichen ersetzt werden, wodurch sich die Lesbarkeit erhöht:

```
$ echo "123456789o123456789o123456789o"
123456789o123456789o123456789o
$ head -5 /etc/group | pr -l1 -e:10
root      x         0
daemon    x         1
bin       x         2
sys       x         3
adm       x         4         pi
```

Im Beispiel wird jeder Doppelpunkt durch so viele Leerzeichen ersetzt, die notwendig sind um auf die nächste 10er Position zu kommen.

Anzeige der wesentlichen Elemente einer Konfigdatei

Als Beispiel dient hier die Konfigurationsdatei zu Tayga, eine NAT64 Implementierung. Die Konfigurationsdatei besteht aus Zeilen mit Schlüsselwörter plus zugeordneten Werten und kann Kommentarzeilen beinhalten. Kommentarzeilen beginnen mit einem "#" am Anfang der Zeile.

Möchte man ausschließlich die relevanten Konfigurationszeilen sehen, filtert man alle Kommentarzeilen weg.

```
$ conffile=/var/tayga/tayga.conf
$ grep -v "^#" $conffile | uniq
$ grep -v "^#" $conffile | grep -v "^[ <TAB>]*$"
```

In der ersten Variante werden alle Kommentarzeilen entfernt, und mehrere aufeinander folgende Leerzeilen (genauer: alle mehrfach auftretenden Zeilen) durch eine einzige ersetzt. Die zweite Form entfernt alle Leerzeilen und sorgt damit für eine noch komprimiertere Darstellung.

Je nach Konfigurationsdatei kann es sinnvoll sein auch Kommentare die mitten in einer Zeile beginnen (und bis zum Ende der Zeile gelten) zu entfernen. Spielt die Reihenfolge der Konfigurationszeilen keine Rolle kann die Ausgabe weiter „normalisiert“ werden in dem man die Zeilen alphabetisch sortiert anordnet. Damit lassen sich dann unterschiedliche Konfigurationsdateien leicht miteinander vergleichen. Zur Not verwendet man *diff* um die Unterschiede herauszuarbeiten.

```
$ grep -v "^#" $conffile | sed "s/#.*$//" |
> grep -v "^[ <TAB>]*$" | sort
data-dir /var/tayga/
dynamic-pool 192.168.64.0/24
ipv4-addr 192.168.64.1
ipv6-addr 2a00:0:1807:64::1
prefix 2a00:0000:1807:0064::/96
tun-device nat64
```

Ein zweites (etwas einfacheres) Beispiel zur Prüfung welche NTP-Server für die Synchronisation des NTP-Zeitdienstes konfiguriert sind:

```
$ egrep "^[   ]*(server|pool)" /etc/ntp.conf
pool 2.pool.ntp.org
server  ntp3.hetzner.net iburst
```

Filtern von Leerzeilen

Entfernt man, wie im obigen Beispiel die Kommentarzeilen aus einer Konfigurationsdatei, bleiben häufig größere Bereiche mit Leerzeilen übrig. Diese können ebenfalls mit *grep*(1) entfernt werden, einfacher und schöner (da eine Leerzeile bleibt) ist es jedoch mit *uniq*(1) alle aufeinaderfolgenden Zeilen durch ein Zeile zu ersetzen.

```
$ grep -v "^#" /etc/ntp.conf | uniq
driftfile /var/lib/ntp/ntp.drift

pool 2.pool.ntp.org

server  ntp3.hetzner.net iburst

restrict -4 default kod notrap nomodify nopeer noquery limited
restrict -6 default kod notrap nomodify nopeer noquery limited

restrict 127.0.0.1
restrict ::1

restrict source notrap nomodify noquery
```

Groß- in Kleinbuchstaben wandeln (oder umgekehrt)

Zur Wandlung von Groß- in Kleinbuchstaben wird das Kommando *tr*(1) benutzt. Damit die Zeichenklassen je nach lokaler Spracheinstellung korrekt umgesetzt werden, muss die Zeichenklasse in eckige Klammern mit Doppelpunkt versehen angegeben werden.

```
$ echo "GROßSCHREIBUNG inkl. UMLAUTE ÄÖÜ" | tr "[:upper:]" "[:lower:]"
großschreibung inkl. umlaute äöü

$ echo "GROßSCHREIBUNG inkl. UMLAUTE ÄÖÜ" | tr "[:lower:]" "[:upper:]"
GROßSCHREIBUNG INKL. UMLAUTE ÄÖÜ
```

Eine Zeile aus einer Datei ausgeben

Das Beispiel ist ein wenig konstruiert, da man in den seltensten Fällen die Zeilennummer der Zeile kennt die man ausgeben möchte. Wie auch immer, das Beispiel zeigt sehr gut, dass es unter Unix häufig unterschiedliche Ansätze gibt, wie ein Problem zu lösen sein könnte.

Im Beispiel soll die 1015te Zeile der Datei `/usr/share/dict/words` ausgegeben werden. Der erste Ansatz basiert auf den beiden Kommandos *head*(1) und *tail*(1). Mit *head*(1) können die ersten *n* Zeilen einer Datei ausgegeben werden. Durch *tail*(1) wird dann davon nur die letzte Zeile ausgegeben.

```
$ head -1015 /usr/share/dict/words | tail -1
Argos
```

Ein anderer Ansatz ist, das Kommando *sed*(1) anstelle von *head*(1) zu verwenden. Das Kommando gibt die Datei auf der Standardausgabe aus und zählt dabei die Zeilen mit. Bei einer bestimmten Zeilennummer kann ein *sed*(1)-Kommando ausgeführt werden. In unserem Falle handelt es sich um `q` für Quit, d.h. die Ausgabe endet mit dieser Zeilennummer.
Anschließend sorgt *tail*(1) erneut dafür, dass nur noch die letzte Zeile übrig bleibt.

```
$ sed 1015q /usr/share/dict/words | tail -1
Argos
```

Durch die unterschiedlichen *sed*(1) Kommandos ergeben sich aber noch andere Möglichkeiten. So sorgt das *sed*(1)-Kommando `p` dafür, dass genau diese Zeile ausgeben wird. Sinn macht dies erst, wenn die default Ausgabe aller Zeilen mit der Option `-n` unterdrückt wird.

```
$ sed -n 1015p /usr/share/dict/words
Argos
```

Der Nachteil des obigen Kommandos ist, dass hierbei immer die gesamte Datei gelesen wird, obwohl ab der 1015ten Zeile klar ist, dass keine weitere Ausgabe mehr erfolgt. Idealerweise möchte man die Kommandos `p`(print) und `q`(uit) kombinieren. Auf der Kommandozeile geht das entweder mit zwei `e`(xpression) Optionen, oder noch einfacher durch das Binden beider Kommandos an die Zeile:

```
$ sed -n -e 1015p -e 1015q /usr/share/dict/words
Argos
$ sed -n '1015{p;q}'  /usr/share/dict/words
Argos
```

Zu guter Letzt noch eine Variante die vor allem einsetzbar ist, wenn die *x*-te nicht-leere Eingabezeile ausgegeben werden soll. Der traditionelle Ansatz ist, zuerst alle Leerzeilen zu entfernen, und dann eines der obigen Kommandos einzusetzen, z.B.

```
$ grep -v "^[ <TAB>]*$" /usr/share/dict/words | sed -n '1015{p;q}'
```

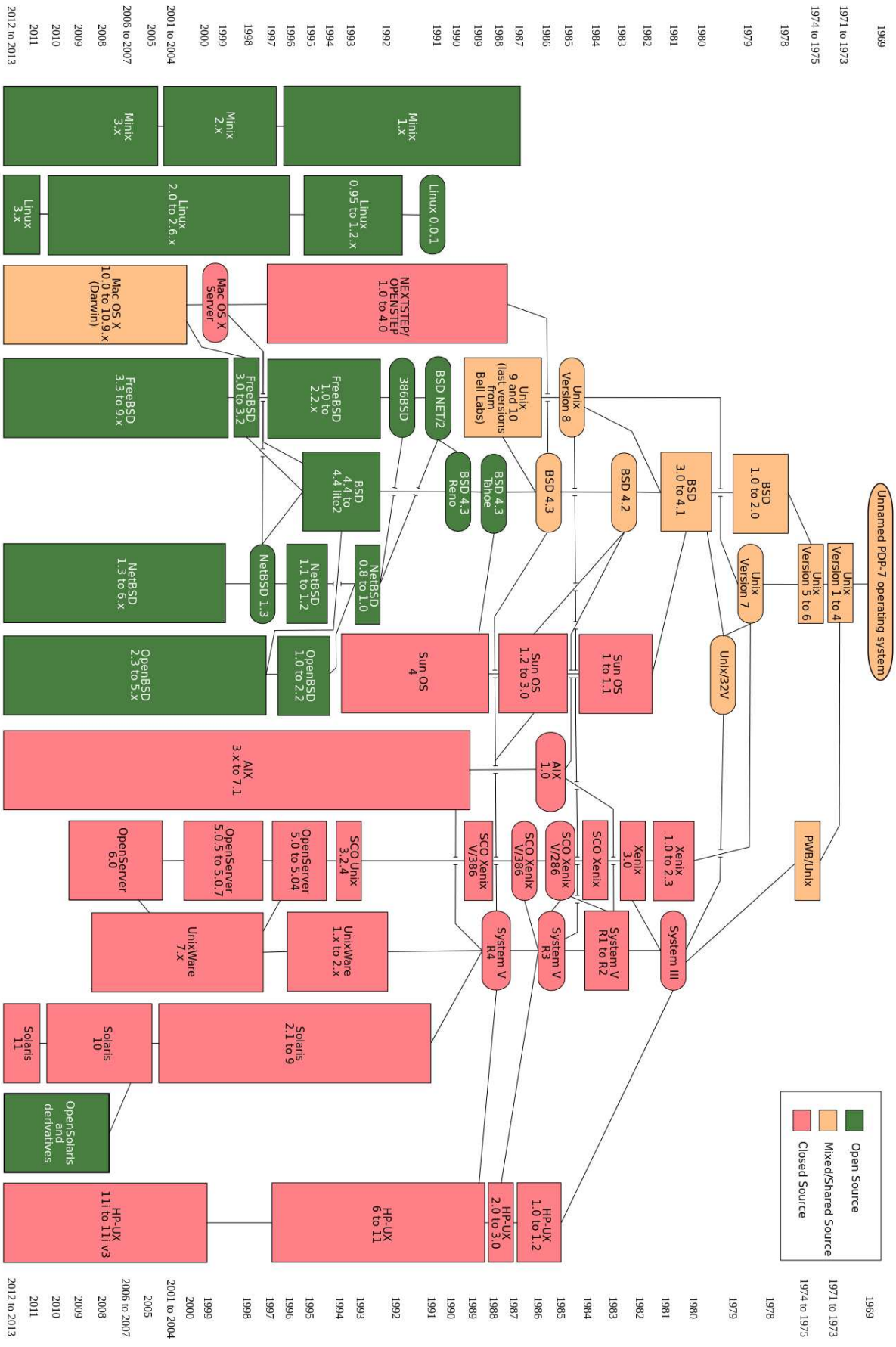
Alternativ wird *nl*(1) verwendet um alle (nicht leeren) Zeilen zu nummerieren. Dann verwenden wir *grep*(1) um die gewünschte Zeile zu finden, und räumen anschließend die Zeilennummer wieder ab:

```
$ nl -w1 -s: /usr/share/dict/words | grep "^1015:" | cut -d":" -f2-
Argos
```

Die Option `-s` von *nl*(1) setzt das Trennzeichen zwischen der Zeilennummer und der eigentlichen Zeile. Durch `-w1` wird die Zeilennummer unformatiert ausgegeben. Beides erleichtert die Definition des Suchmusters und das anschliessende Abräumen.

A. UNIX timeline

https://en.wikipedia.org/wiki/List_of_Unix_systems#/media/File:Unix_history-simple.svg



B. Stichwortverzeichnis

D

E

F

G

H

I

J

K

L

M

N

O

P

W

X

Z

C. Literaturhinweise

[AKW88]

Aho Alfred V., Kernighan Brian W., Weinberger Peter J.
The AWK Programming Language
Addison Wesley 1988

Das Standardwerk zu `awk`, geschrieben von den Autoren des Programms (sozusagen die AWK Bibel).

[AT&T88]

AT & T
Programmer's Reference Manual
System Administrator's Reference Manual
User's Reference Manual
Prentice-Hall 1988 (deutsch bei Hanser)

Wer keine Dokumentation hat, oder sich mit der englischen Beschreibung schwertut, findet hier die (schlechte) Übersetzung der Original(!) AT&T Dokumentation. Die Bücher enthalten daher einige Kommandos, die man nur bei AT&T Unix findet. Hat man ein anderes Unix System ist zumindest mit Abweichungen zu rechnen. Trotzdem empfehlenswert z.B. der Band mit den Beschreibungen der C-Funktionen.

[B&D86]

Bach F., Domann P.
Unix Tabellenbuch
Hanser 1986

Tabellarische Gegenüberstellung der Kommandos diverser Unix Versionen. Nur interessant, wenn man ein Buch über Unix schreiben möchte.

[Bac86]

Bach Maurice J.
The design of the UNIX Operating System
Prentice-Hall 1986

Alles über den internen Aufbau von Unix. Schwere Kost. Wirklich nur für diejenigen interessant, die es ganz genau wissen wollen bzw. müssen.
1991 erschien im Hanser Verlag die deutsche Übersetzung („UNIX – Wie funktioniert das Betriebsystem").

[B&K91]

Bolsky Morris L., Korn David G.
Die KornShell
Beschreibung und Referenzhandbuch zur Befehls- und Programmiersprache.
Mc Graw-Hill 1987

Standardwerk zur `ksh`. Geht sehr stark auf die Möglichkeiten bei der Programmierung der KornShell ein. Wenn man portable Batchdateien schreiben will, nutzt man die Möglichkeiten der Programmierung der KornShell sehr wenig. Mir wäre daher eine ausführlichere Beschreibung der interaktiven Möglichkeiten lieber gewesen.

[Bol88]

Bolsky Morris L.
UNIX Text Editor
Das vi Handbuch
Hanser 1988

DAS Nachschlagewerk für den `vi`.

[Fox85]

Foxley Eric
Unix for Super Users
Addison-Wesley 1985

Das einzige Buch, das speziell auf die Systemverwaltertätigkeiten eingeht. Leider ist die Systemverwaltung stark von der Version bzw. dem Hersteller des Unix Systems abhängig. Trotzdem eine gute Einführung in die Systemverwaltung.
Ist 1987 in deutscher Sprache erschienen.

[Gul88]

Gulbins Juergen
UNIX Version 7 bis System V.3
Springer 1988

Der Klassiker. Ein Rundumschlag. Beschreibt auch (ansatzweise) die Unterschiede zwischen den einzelnen UNIX Versionen. Leider strotzt das Buch — trotz 3. (überarbeiteter ?) Auflage — vor Fehlern (insb. bei der Kommandosyntax).

[K&P86]

Kernighan B. W., Pike R.
Der Unix Werkzeugkasten
Hanser 1986

Eins der ersten deutschsprachigen Bücher über Unix. Es beschreibt leider die etwas veraltete Version 7. Trotzdem das beste Unix Buch überhaupt, da es das Zusammenwirken der einzelnen Tools (shell, sed, awk, C, make, Textverarbeitung) sehr gut darstellt. Obwohl es als Anfängerbuch geschrieben ist, sind C-Kenntnisse Voraussetzung und unter Unix sollte man auch schon das eine oder andere kennen.

[Roc85]

Rochkind M. J.
Unix Programmierung für Fortgeschrittene
Hanser 1985

Das Standardwerk für diejenigen, die unter Unix C-Programme schreiben. Enthält die Beschreibung der C Betriebsystemschnittstelle. C-Kenntnisse sind absolute Voraussetzung.

[Sch87]

Schreiner Axel T.
Prof. Schreiner's UNIX - Sprechstunde
Hanser 1987

Tips und Tricks rund um Unix, C-Programmierung, Textverarbeitung, viel awk, und ein wenig über binäre Bäume. Kein UNIX Lehrbuch; es zeigt UNIX eher als Betriebsystem für die Programmentwicklung.

