

Werkzeuge und Arbeitstechniken der Computerlinguistik

Matthias Bethke

`bethke@linguistik.uni-erlangen.de`

Linguistische Informatik
Universität Erlangen-Nürnberg

Wintersemester 2006/2007

Organisatorisches

- 1 Termine, Belegung, etc.

Organisatorisches

- ① Termine, Belegung, etc.
- ② WWW: `http://www.linguistik.uni-erlangen.de/~msbethke/teaching/WS2006-07-WACL.html`

Organisatorisches

- 1 Termine, Belegung, etc.
- 2 WWW: `http://www.linguistik.uni-erlangen.de/~msbethke/teaching/WS2006-07-WACL.html`
- 3 Abschlussklausur

Kursübersicht

- Einführung in Unix/Linux
 - Die Kommandozeile
 - Dateien und Verzeichnisse
 - Befehle zur Dateimanipulation, Textverarbeitung etc.
 - Netzwerkdienste

Kursübersicht

- Einführung in Unix/Linux
 - Die Kommandozeile
 - Dateien und Verzeichnisse
 - Befehle zur Dateimanipulation, Textverarbeitung etc.
 - Netzwerkdienste
- Texteditoren: *Emacs* und *vi*

Kursübersicht

- Einführung in Unix/Linux
 - Die Kommandozeile
 - Dateien und Verzeichnisse
 - Befehle zur Dateimanipulation, Textverarbeitung etc.
 - Netzwerkdienste
- Texteditoren: *Emacs* und *vi*
- Das Textsatzsystem TeX/LaTeX

Kursübersicht

- Einführung in Unix/Linux
 - Die Kommandozeile
 - Dateien und Verzeichnisse
 - Befehle zur Dateimanipulation, Textverarbeitung etc.
 - Netzwerkdienste
- Texteditoren: *Emacs* und *vi*
- Das Textsatzsystem TeX/LaTeX
- Reguläre Ausdrücke

Kursübersicht

- Einführung in Unix/Linux
 - Die Kommandozeile
 - Dateien und Verzeichnisse
 - Befehle zur Dateimanipulation, Textverarbeitung etc.
 - Netzwerkdienste
- Texteditoren: *Emacs* und *vi*
- Das Textsatzsystem TeX/LaTeX
- Reguläre Ausdrücke
- Markupssprachen, insbes. HTML und XML

Benutzerkennungen/Accounts

- Benutzername: *erster* und *letzer* Buchstabe des Vornamens + *erste sechs* Buchstaben des Nachnamens.
Deutsche Umlaute per Umschrift, Akzente weggelassen.
z. B.: Carsten Müller → cnmuelle
 Ándrea Březový → aabrezov
 Li Qi → liqi
- Anfangspasswort „alpha“

Benutzerkennungen/Accounts

- Benutzername: *erster* und *letzer* Buchstabe des Vornamens + *erste sechs* Buchstaben des Nachnamens.
Deutsche Umlaute per Umschrift, Akzente weggelassen.
z. B.: Carsten Müller → cnmuelle
Ándrea Březový → aabrezov
Li Qi → liqi
- Anfangspasswort „alpha“
- Verschiedene Desktops zur Auswahl auf dem Login-Bildschirm: KDE, XFCE, Gnome, fvwm2, etc.

Benutzerkennungen/Accounts

- Benutzername: *erster* und *letzer* Buchstabe des Vornamens + *erste sechs* Buchstaben des Nachnamens.
Deutsche Umlaute per Umschrift, Akzente weggelassen.

z. B.: Carsten Müller → cnmuelle

Ándrea Březový → aabrezov

Li Qi → liqi

- Anfangspasswort „alpha“

 - Verschiedene Desktops zur Auswahl auf dem Login-Bildschirm: KDE, XFCE, Gnome, fvwm2, etc.
- Desktop auswählen und einloggen!


Erste Schritte: Passwort

- **Passwort ändern!**

- Terminalapplikation öffnen



Piktogramm sieht so ähnlich aus:

- Kommando **passwd** eingeben und mit  abschließen
- *Aktuelles* Passwort wird abgefragt.
- Anschließend wird *zweimal* das *neue* Passwort abgefragt (zur Vermeidung von Tippfehlern bei verdeckter Eingabe).


Erste Schritte: Passwort

- **Passwort ändern!**

- Terminalapplikation öffnen



Piktogramm sieht so ähnlich aus:

- Kommando **passwd** eingeben und mit  abschließen
 - *Aktuelles* Passwort wird abgefragt.
 - Anschließend wird *zweimal* das *neue* Passwort abgefragt (zur Vermeidung von Tippfehlern bei verdeckter Eingabe).
- Ausprobieren: ausloggen (normalerweise im „Start-Menü“ unten links) und mit neuem Passwort wieder einloggen.

Erste Schritte: wichtige Applikationen

- Email
 - KMail (KDE Mailprogramm)
 - Thunderbird (Ex-Mozilla)
 - Mutt (Textmodus)
- WWW
 - Firefox
 - SeaMonkey
 - Konqueror (KDE-Browser, dient auch als Filemanager)
- Terminals
 - Gnome-Terminal
 - Konsole

Bitte ausprobieren: funktioniert das alles? Ist die CLUE-Homepage aufrufbar? Bekomme ich Mail?

Die Kommandozeile (1)

Die meisten Arbeiten in diesem Kurs finden auf der Kommandozeile statt.

Die Kommandozeile (1)

Die meisten Arbeiten in diesem Kurs finden auf der Kommandozeile statt.

Warum Kommandozeile?

- Grafische Schnittstellen für grafische Inhalte, textuelle Schnittstellen für Texte. Wir verarbeiten Texte.
- Extrem ausdrucksstark
- Oft schneller und effizienter als rumklicken
- Fließender Übergang zur „richtigen Programmierung“; Automatisierung von häufigen Aufgaben

Die Kommandozeile (2)



- Ein Terminal präsentiert sich als Fenster oder ganzer Textbildschirm¹ mit einem *Prompt* („Eingabeaufforderung“) und üblicherweise einem *Cursor*.

¹Ausprobieren: **Strg** + **Alt** + **F1** – zurück mit **Strg** + **Alt** + **F7**

²Aber nach Belieben änderbar, bei SuSE z. B. „>“

Die Kommandozeile (2)



- Ein Terminal präsentiert sich als Fenster oder ganzer Textbildschirm¹ mit einem *Prompt* („Eingabeaufforderung“) und üblicherweise einem *Cursor*.
- Im Terminal läuft die sog. *Shell*, der Kommandointerpreter. Unter Linux ist dies meist die *bash* („Bourne Again Shell“)

¹Ausprobieren: **Strg** + **Alt** + **F1** – zurück mit **Strg** + **Alt** + **F7**

²Aber nach Belieben änderbar, bei SuSE z. B. „>“

Die Kommandozeile (2)



- Ein Terminal präsentiert sich als Fenster oder ganzer Textbildschirm¹ mit einem *Prompt* („Eingabeaufforderung“) und üblicherweise einem *Cursor*.
- Im Terminal läuft die sog. *Shell*, der Kommandointerpreter. Unter Linux ist dies meist die *bash* („Bourne Again Shell“)
- Der Prompt zeigt an, dass Befehle oder Daten eingegeben werden können und sieht je nach Kontext unterschiedlich aus.
- Charakteristisch² für den Befehls-Prompt der Bash: das Dollarzeichen

¹Ausprobieren: **[Strg]** + **[Alt]** + **[F1]** – zurück mit **[Strg]** + **[Alt]** + **[F7]**

²Aber nach Belieben änderbar, bei SuSE z. B. „>“

Befehle und Argumente (1)

Beispiel

```
msbethke@clue45 ~ $ ls -l Documents/
```

Befehle und Argumente (1)

Beispiel

```
msbethke@clue45 ~ $ ls -l Documents/
```

- Als erstes am Prompt wird ein *Befehl* (hier: „**ls**“) eingegeben.
- **Vorsicht:** Groß- und Kleinschreibung werden unterschieden!

Befehle und Argumente (1)

Beispiel

```
msbethke@clue45 ~ $ ls -l Documents/
```

- Als erstes am Prompt wird ein *Befehl* (hier: „**ls**“) eingegeben.
- **Vorsicht:** Groß- und Kleinschreibung werden unterschieden!
- Folgen auf den Befehl noch weitere Eingaben, werden sie mit Leerzeichen davon getrennt. Hier: „**-l Documents**“.
- Auf den Befehl folgende Eingaben auf der Kommandozeile heißen *Argumente*. Einzelne Argumente werden ebenfalls mit Leerzeichen getrennt.

Befehle und Argumente (1)

Beispiel


```
msbethke@clue45 ~ $ ls -l Documents/
```

- Als erstes am Prompt wird ein *Befehl* (hier: „**ls**“) eingegeben.
- **Vorsicht:** Groß- und Kleinschreibung werden unterschieden!
- Folgen auf den Befehl noch weitere Eingaben, werden sie mit Leerzeichen davon getrennt. Hier: „**-l Documents**“.
- Auf den Befehl folgende Eingaben auf der Kommandozeile heißen *Argumente*. Einzelne Argumente werden ebenfalls mit Leerzeichen getrennt.
- Ein Argument, das mit einem speziellen Zeichen beginnt (unter Unix meist ein *einfacher* oder *doppelter Bindestrich*), heißt *Option* und beeinflusst das Verhalten des Befehls.

Befehle und Argumente (1)

Beispiel

```
msbethke@clue45 ~ $ ls -l Documents/
```

- Als erstes am Prompt wird ein *Befehl* (hier: „**ls**“) eingegeben.
- **Vorsicht:** Groß- und Kleinschreibung werden unterschieden!
- Folgen auf den Befehl noch weitere Eingaben, werden sie mit Leerzeichen davon getrennt. Hier: „**-l Documents**“.
- Auf den Befehl folgende Eingaben auf der Kommandozeile heißen *Argumente*. Einzelne Argumente werden ebenfalls mit Leerzeichen getrennt.
- Ein Argument, das mit einem speziellen Zeichen beginnt (unter Unix meist ein *einfacher* oder *doppelter Bindestrich*), heißt *Option* und beeinflusst das Verhalten des Befehls.
- Erst beim drücken der Eingabetaste  wird der Befehl ausgeführt.

Befehle und Argumente (2)

Befehle, Optionen und andere Argumente lassen sich grob mit Verben, Adjektiven und Substantiven in einem Satz vergleichen:

- Befehle \Leftrightarrow Verben: *was* soll getan werden?
- Optionen \Leftrightarrow Adjektive: *wie* soll es getan werden?
- Andere Argumente \Leftrightarrow Substantive: und *mit was*?

Befehle und Argumente (2)

Befehle, Optionen und andere Argumente lassen sich grob mit Verben, Adjektiven und Substantiven in einem Satz vergleichen:

- Befehle \Leftrightarrow Verben: *was* soll getan werden?
- Optionen \Leftrightarrow Adjektive: *wie* soll es getan werden?
- Andere Argumente \Leftrightarrow Substantive: und *mit was*?

Folgerung

Der Benutzer kommuniziert mit der Unix-Shell in einer Art *formaler Sprache* mit ihrer eigenen (relativ primitiven) Syntax.

Befehle und Argumente (2)

Befehle, Optionen und andere Argumente lassen sich grob mit Verben, Adjektiven und Substantiven in einem Satz vergleichen:

- Befehle \Leftrightarrow Verben: *was* soll getan werden?
- Optionen \Leftrightarrow Adjektive: *wie* soll es getan werden?
- Andere Argumente \Leftrightarrow Substantive: und *mit was*?

Folgerung

Der Benutzer kommuniziert mit der Unix-Shell in einer Art *formaler Sprache* mit ihrer eigenen (relativ primitiven) Syntax.

Übersetzung des vorigen Beispiels

```
msbethke@clue45 ~ $ ls -l Documents/
```

ls: liste-auf

-l: lang

Documents/: das Verzeichnis namens „Documents“

Interludium: typographische Konventionen

Ich versuche mich im Folgenden an typographische Konventionen zu halten, wie sie einiger technischer Literatur üblich sind:

- Kursivschrift:* Namen von Dateien, Verzeichnissen, Benutzern etc. Außerdem zur Hervorhebung neu eingeführter Begriffe.
- Schreibmaschine: Ausgaben des Systems
- Schreibmaschine kursiv:* In Beispielen für generische Bezeichnungen, z. B. „*Dateiname*“
→ „hier einen Dateinamen eingeben“
- Schreibmaschine fett:** Text, der genau so eingegeben werden muss; außerdem zur Hervorhebung in Beispielen.

Der `ls`-Befehl und seine Optionen

Dieser Befehl listet Dateien und Verzeichnisse auf und gibt verschiedene Informationen dazu. Jedes Argument, das keine Option ist, wird als Name eines Verzeichnisses bzw. einer Datei interpretiert. Ohne Argument bzw. mit ausschließlich Optionen wird das aktuelle Verzeichnis angezeigt.

- l (long): langes Ausgabeformat mit Details zu jedem Eintrag
- a (all): auch Dateien zeigen, deren Name mit einem Punkt beginnt
- R (recursive): alle Dateien und Verzeichnisse in allen beliebig tief geschachtelten Unterverzeichnissen der Argumente auflisten
- t (time): nach Zeitpunkt der letzten Änderung sortieren
- r (reverse): Sortierreihenfolge umkehren
- d (directory): Als Argument angegebene Verzeichnisse mit ihrem Namen anzeigen, nicht ihren Inhalt

...und woher weiß ich sowas?!

- *Befehl* **--help**
 - Wird von den meisten Befehlen (zumindest unter Linux) bereitgestellt und gibt eine kurze Hilfe aus.
 - Häufiges Synonym: **-h**
 - Falls weder das eine noch das andere funktioniert: oft gibt es Hilfe beim Starten eines Befehls mit falschen Argumenten.

...und woher weiß ich sowas?!

- *Befehl* **--help**
 - Wird von den meisten Befehlen (zumindest unter Linux) bereitgestellt und gibt eine kurze Hilfe aus.
 - Häufiges Synonym: **-h**
 - Falls weder das eine noch das andere funktioniert: oft gibt es Hilfe beim Starten eines Befehls mit falschen Argumenten.
- **man** *Befehl*
 - Gibt den Handbucheintrag („Manual“) zum angegebenen Befehl aus.
 - Nach Stichworten in Handbuch-Überschriften suchen:
man -k *Stichwort* („keyword“) oder synonym: **apropos**.
 - Nach Stichworten im *kompletten* Handbuch suchen:
man -K *Stichwort*

Dateien und Verzeichnisse

Dateien und Verzeichnisse

- Eine Datei ist eine *Sammlung zusammengehöriger Daten*, die auf einem Speichermedium abgelegt ist.

Dateien und Verzeichnisse

- Eine Datei ist eine *Sammlung zusammengehöriger Daten*, die auf einem Speichermedium abgelegt ist.
- Dateien, die nicht das gesamte Medium belegen, benötigen einen *Namen*, der sie identifiziert und auffindbar macht.
- Gewöhnlich gehören zu einer Datei auch *Attribute* bzw. *Metadaten* wie die Art der Datei, ihr Besitzer, Berechtigungen zum Lesen, Schreiben, Ausführen, Löschen etc. und ihre Größe.

Dateien und Verzeichnisse

- Eine Datei ist eine *Sammlung zusammengehöriger Daten*, die auf einem Speichermedium abgelegt ist.
- Dateien, die nicht das gesamte Medium belegen, benötigen einen *Namen*, der sie identifiziert und auffindbar macht.
- Gewöhnlich gehören zu einer Datei auch *Attribute* bzw. *Metadaten* wie die Art der Datei, ihr Besitzer, Berechtigungen zum Lesen, Schreiben, Ausführen, Löschen etc. und ihre Größe.
- Ein Verzeichnis ist eine spezielle Datei, die Namen und andere Metadaten (s. o.) über andere Dateien enthält.

Dateien und Verzeichnisse

- Eine Datei ist eine *Sammlung zusammengehöriger Daten*, die auf einem Speichermedium abgelegt ist.
 - Dateien, die nicht das gesamte Medium belegen, benötigen einen *Namen*, der sie identifiziert und auffindbar macht.
 - Gewöhnlich gehören zu einer Datei auch *Attribute* bzw. *Metadaten* wie die Art der Datei, ihr Besitzer, Berechtigungen zum Lesen, Schreiben, Ausführen, Löschen etc. und ihre Größe.
 - Ein Verzeichnis ist eine spezielle Datei, die Namen und andere Metadaten (s. o.) über andere Dateien enthält.
 - Ein Verzeichnis bildet einen *Namensraum*, innerhalb dessen jeder Name eindeutig sein muss.
- Innerhalb einer Hierarchie von Verzeichnissen ist jede Datei eindeutig durch ihren *Pfad* identifiziert.

Pfade

Ein „Pfad“ zum Rechner clue14 in der realen Welt

/Deutschland/Erlangen-91054/Bismarckstraße/12/0.319/clue14

Pfade

Ein „Pfad“ zum Rechner clue14 in der realen Welt

/Deutschland/Erlangen-91054/Bismarckstraße/12/0.319/clue14

- Pfadbestandteile sind mit Schrägstrichen von einander getrennt.

Pfade

Ein „Pfad“ zum Rechner clue14 in der realen Welt

/Deutschland/Erlangen-91054/Bismarckstraße/12/0.319/clue14

- Pfadbestandteile sind mit Schrägstrichen von einander getrennt.
 - Vorangestellter Schrägstrich → „ganz oben anfangen“
- „Relative Pfade“ sind möglich

Pfade

Ein „Pfad“ zum Rechner clue14 in der realen Welt

```
/Deutschland/Erlangen-91054/Bismarckstraße/12/0.319/clue14
```

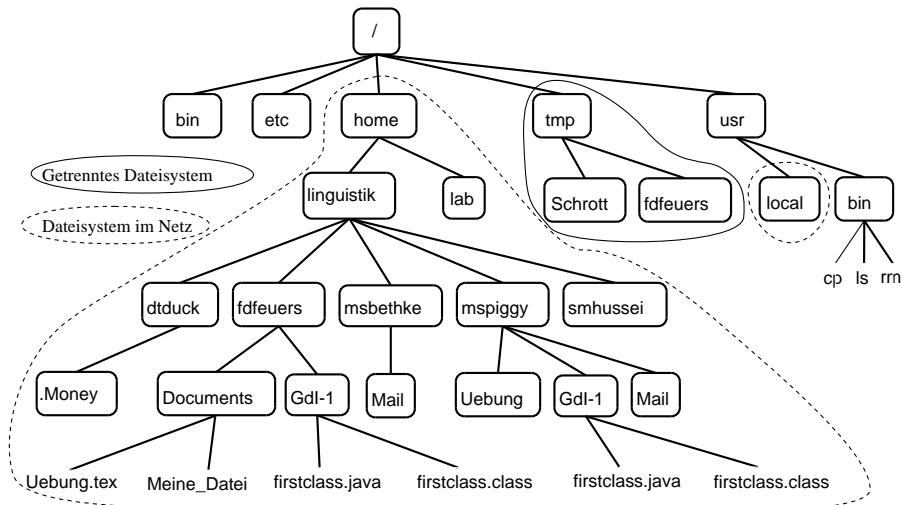
- Pfadbestandteile sind mit Schrägstrichen von einander getrennt.
 - Vorangestellter Schrägstrich → „ganz oben anfangen“
- „Relative Pfade“ sind möglich

Relativer „Pfad“ zu clue14

```
../0.319/clue14
```

- Der Name „..“ hat hier die vordefinierte Bedeutung „eine Ebene nach oben“!
- Ebenso bedeutet „.“ die aktuelle Ebene. Diese Namen sind unter Unix fest vergeben und keine echte Datei darf so heißen.

Ein Unix-Verzeichnisbaum



Aktuelles Verzeichnis / Bewegen im Verzeichnisbaum

- Jedes Programm hat ein „aktuelles Verzeichnis“
 - Die Bash zeigt das aktuelle (oder *Arbeitsverzeichnis*) i. d. R. im Prompt an.
 - Relative Pfadangaben in einem Programm beziehen sich grundsätzlich auf dieses Verzeichnis.
 - Wenn es nicht explizit anders angegeben wurde, nehmen viele Programme als anfängliches Arbeitsverzeichnis das *Home-Verzeichnis* des jeweiligen Benutzers.
 - Das Kürzel „`~`“ (Tilde) bezieht sich in der Shell immer auf das Home-Verzeichnis
- Der Befehl `cd Pfad` („change directory“) wechselt das Arbeitsverzeichnis. Ohne Argument wechselt er ins Home-Verzeichnis.
- Der Befehl `pwd` („print working directory“) gibt das aktuelle Verzeichnis aus.

Allgemeines zu Datei- und Verzeichnisnamen

- Maximale Länge üblicherweise je 255 Zeichen
- Darf so gut wie alle Zeichen außer dem Null-Zeichen und dem Pfadtrenner (Schrägstrich) enthalten; aus Kompatibilitätsgründen sollte man sich aber auf folgende beschränken:
 - Klein- und Großbuchstaben A-Z
 - Ziffern
 - Punkt, Bindestrich, Unterstrich (-)
 - Eventuell Umlaute

Allgemeines zu Datei- und Verzeichnisnamen

- Maximale Länge üblicherweise je 255 Zeichen
- Darf so gut wie alle Zeichen außer dem Null-Zeichen und dem Pfadtrenner (Schrägstrich) enthalten; aus Kompatibilitätsgründen sollte man sich aber auf folgende beschränken:
 - Klein- und Großbuchstaben A-Z
 - Ziffern
 - Punkt, Bindestrich, Unterstrich (-)
 - Eventuell Umlaute
- Probleme mit anderen Zeichen
 - Leerzeichen dienen der Trennung von Argumenten voneinander.
 - Viele Sonderzeichen (Doppelkreuz, Dollar, Kaufmannsund, div. Klammern etc.) etc. haben spezielle Bedeutung in der Shell.
 - Umlaute werden nicht auf allen Systemen gleich kodiert und können z. B. beim Wechsel an andere Rechner am gleichen Server oder beim Versand per Email Probleme bereiten.

Detaillierte Ausgabe von `ls`

Die „lange“ Ausgabe von `ls3` enthält die meisten Metadaten, die im Unix-Dateisystem zu einer Datei verfügbar sind.

Die Ausgabe von `ls -l`

```
-r----- 1 msbethke users 23024 2006-03-06 14:44 A1201.pdf
-rw----- 1 msbethke users 725 2005-07-29 17:28 AdminDay
-rw----- 1 msbethke users 633180 2006-02-24 01:29 alderm.pdf
-rw----- 1 msbethke users 15967 2005-10-18 12:53 alternate.pdf
-rwxr-xr-x 1 msbethke users 6540 2006-08-04 22:53 a.out
-rw-rw-r-- 1 msbethke users 26802 2005-01-23 19:23 arbeit.html
drwx----- 11 msbethke users 4096 2006-10-27 17:09 Mail
```

³Tip: die meisten Systeme definieren `ll` und/oder `dir` bereits als *alias* (Abkürzung) für `ls -l`!

Befehle *noch und nöcher* (1)

Die folgenden grundlegenden Dateimanipulationsbefehle werden wir im Folgenden ausprobieren:

`rm Datei...` : Datei löschen

`mkdir Verzeichnis...` Verzeichnis anlegen

`rmdir Verzeichnis...` Verzeichnis löschen

`mv Quelle... Ziel` Dateien oder Verzeichnisse umbenennen
oder an eine andere Stelle verschieben

`cp Quelle ... Ziel` Dateien oder Verzeichnisse⁴ kopieren.

`less Datei ...` Dateien seitenweise auf dem Terminal
anzeigen

⁴Hierfür wird die Option `-R` benötigt!

Befehle *noch und nöcher* (2)

- chmod** *Modus Datei...* Berechtigungen von Dateien entsprechend *Modus* (vgl. Man-Page!) ändern
- cat** *Datei...* Dateien ausgeben (hauptsächlich zur Umleitung in andere Dateien/Befehle)
- mail** *Adresse* Email verschicken (Ende mit Punkt am Zeilenanfang oder **Strg** + **D**)
- gzip** *Datei...* Datei komprimieren (dekomprimieren mit Option **-d** oder Befehl **gunzip**)

Wildcards

Um Dateien oder Verzeichnisse nicht immer explizit mit allen Namen angeben zu müssen, versteht die Shell sog. *Wildcards*. Diese Platzhalter stehen für verschiedene Zeichen, ein Name mit Wildcards „passt“ also i. d. R. auf mehrere Dateinamen:

? Eine beliebiges Zeichen

z. B. `ls .?*`: alle Dateien/Verzeichnisse auflisten, deren Name mit einem Punkt beginnt und genau zwei Zeichen lang ist

* Eine beliebige Anzahl beliebiger Zeichen

z. B. `ls A*`: alles auflisten, was mit großem A beginnt

[...] Irgendeins der zwischen den Klammern aufgelisteten Zeichen.

Auch Bereiche wie „a-z“ können angegeben werden.

z. B. `rm [Ab-d]*`: alle Dateien löschen, die mit großem A, kleinem b, c oder d beginnen

Übung (1)

- Legen Sie im Verzeichnis */tmp* ein Verzeichnis an, dessen Name Ihrem Benutzernamen entspricht
- Informieren Sie sich im Systemhandbuch, was der Parameter *Modus* des Befehls **chmod** bewirkt.
- Ändern Sie die Berechtigungen des Verzeichnisses so, dass
 - Sie selbst lesen, schreiben und ausführen (=Verzeichnis betreten)
 - Benutzer Ihrer Gruppe zwar lesen aber nicht ausführen oder schreiben
 - Andere ausführen aber weder lesen noch schreiben dürfen
- Legen Sie ein Verzeichnis *Korpus* an, kopieren Sie die Datei */korpora/Limas/limas.sentences* in dieses Verzeichnis und benennen Sie sie in *limas* um. (Wie geht das in einem Schritt? Und in zwei?)

Übung (2)

- Schauen Sie sich die Dateigröße und Berechtigungen an. Komprimieren Sie die Datei und vergleichen Sie.
- Versuchen Sie, die Datei mit `less` anzuzeigen.
- Verlassen Sie das Verzeichnis *Korpus* und versuchen Sie, es mit `rmdir` bzw. `rm` zu löschen. Unter welchen Bedingungen geht das?
- Verlassen Sie Ihr Verzeichnis in `/tmp` und löschen Sie es.
- Versuchen Sie, in das Home-Verzeichnis von *msbethke* zu wechseln und sich den Inhalt anzeigen zu lassen. Was passiert?
- Schauen Sie sich die Liste der Linguistik-Homeverzeichnisse an. Wie unterscheiden sie sich bezüglich des zuletzt aufgetretenen Problems?

Ein-/Ausgabekanäle: Grundlagen

- Jedes Unix-Programm⁵ hat drei *Standard-IO*⁶-Ströme
 - Eingabe (*stdin*, Kanal 0)
 - Ausgabe (*stdout*, Kanal 1)
 - Fehlerkanal (*stderr*, Kanal 2)

⁵Das gilt auch für die meisten anderen Betriebssysteme

⁶*Input/Output* ~ Ein-/Ausgabe

Ein-/Ausgabekanäle: Grundlagen

- Jedes Unix-Programm⁵ hat drei *Standard-IO*⁶-Ströme
 - Eingabe (*stdin*, Kanal 0)
 - Ausgabe (*stdout*, Kanal 1)
 - Fehlerkanal (*stderr*, Kanal 2)
- Normalerweise sind diese Ströme alle mit einem Terminal verbunden, sie lassen sich aber in der Shell einzeln umleiten.
- Eingaben für Befehle können aus einer Datei oder einem anderen Befehl kommen, Ausgaben in eine Datei oder einen anderen Befehl gehen.

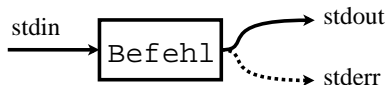


Abbildung: Das Filterkonzept unter Unix

⁵Das gilt auch für die meisten anderen Betriebssysteme

⁶*Input/Output* ~ Ein-/Ausgabe

Ein-/Ausgabekanäle: Umleitung (1)

- Ein- und Ausgabeströme werden mit Kleiner- bzw. Größer-Zeichen in Dateien umgeleitet (Mnemonic: „Pfeil in der Richtung, in der Daten fließen“)

Ein-/Ausgabekanäle: Umleitung (1)

- Ein- und Ausgabeströme werden mit Kleiner- bzw. Größer-Zeichen in Dateien umgeleitet (Mnemonic: „Pfeil in der Richtung, in der Daten fließen“)
- Ausgabeumleitung in eine Datei: *Befehl > Dateiname*
z. B. `ls -l > dateiliste`
- Eingabeumleitung aus einer Datei: *Befehl < Dateiname*
z. B. `less < dateiliste`
- Fehlerumleitung in eine Datei: *Befehl 2> Dateiname*
z. B. `ls -l ~ /gipsnich 2> fehler`

Ein-/Ausgabekanäle: Umleitung (1)

- Ein- und Ausgabeströme werden mit Kleiner- bzw. Größer-Zeichen in Dateien umgeleitet (Mnemonic: „Pfeil in der Richtung, in der Daten fließen“)
- Ausgabeumleitung in eine Datei: *Befehl > Dateiname*
z. B. `ls -l > dateiliste`
- Eingabeumleitung aus einer Datei: *Befehl < Dateiname*
z. B. `less < dateiliste`
- Fehlerumleitung in eine Datei: *Befehl 2> Dateiname*
z. B. `ls -l ~ /gipsnich 2> fehler`

Vorsicht!

Wenn die Datei rechts vom Umleitungspfeil schon existiert, wird sie überschrieben! Der Inhalt ist *nicht^a wieder herstellbar!*

^aAbgesehen von Backups

Ein-/Ausgabekanäle: Umleitung (2)

- Soll die Programmausgabe an eine Datei angehängt werden, statt sie zu überschreiben, können *zwei Größer-Zeichen* statt einem verwendet werden:
z. B. `pwd >> ~/Verzeichnisse`

Ein-/Ausgabekanäle: Umleitung (2)

- Soll die Programmausgabe an eine Datei angehängt werden, statt sie zu überschreiben, können *zwei Größer-Zeichen* statt einem verwendet werden:
z. B. `pwd >> ~/Verzeichnisse`
- Ausgabeumleitung allgemein:

Strom→Datei, überschreiben: *Befehl [Strom]>Datei*

Strom→Datei, anhängen: *Befehl [Strom]>>Datei*

Strom→Strom: *Befehl [Strom]>&Strom*

Ein-/Ausgabekanäle: Umleitung (2)

- Soll die Programmausgabe an eine Datei angehängt werden, statt sie zu überschreiben, können *zwei Größer-Zeichen* statt einem verwendet werden:

z. B. `pwd >> ~/Verzeichnisse`

- Ausgabeumleitung allgemein:

Strom→Datei, überschreiben: `Befehl [Strom]>Datei`

Strom→Datei, anhängen: `Befehl [Strom]>>Datei`

Strom→Strom: `Befehl [Strom]>&Strom`

- Vor dem Dateinamen kann ein Leerzeichen stehen, nicht jedoch vor einer Kanalnummer (s. u.)!

Ein-/Ausgabekanäle: *Pipes*

- Umleitungen von/in andere Befehle funktionieren über *Pipes*
- Allgemeines Format: *Befehl*₁ | *Befehl*₂
→ *Befehl*₂ bekommt die *stdout*-Ausgaben von *Befehl*₁ auf *stdin*
- Leerzeichen links und rechts des Pipe-Symbols sind erlaubt, aber nicht unbedingt notwendig.

Ein-/Ausgabekanäle: *Pipes*

- Umleitungen von/in andere Befehle funktionieren über *Pipes*
- Allgemeines Format: $Befehl_1 \mid Befehl_2$
→ $Befehl_2$ bekommt die *stdout*-Ausgaben von $Befehl_1$ auf *stdin*
- Leerzeichen links und rechts des Pipe-Symbols sind erlaubt, aber nicht unbedingt notwendig.
- Beliebig viele Befehle können in einer solchen „Pipeline“ hintereinandergeschaltet werden.
- Die Befehle werden alle auf einmal gestartet, d. h. $Befehl_n$ kann schon Daten verarbeiten, bevor $Befehl_{n-1}$ fertig ist

Ein-/Ausgabekanäle: *Pipes*

- Umleitungen von/in andere Befehle funktionieren über *Pipes*
- Allgemeines Format: *Befehl*₁ | *Befehl*₂
→ *Befehl*₂ bekommt die *stdout*-Ausgaben von *Befehl*₁ auf *stdin*
- Leerzeichen links und rechts des Pipe-Symbols sind erlaubt, aber nicht unbedingt notwendig.
- Beliebig viele Befehle können in einer solchen „Pipeline“ hintereinandergeschaltet werden.
- Die Befehle werden alle auf einmal gestartet, d. h. *Befehl*_n kann schon Daten verarbeiten, bevor *Befehl*_{n-1} fertig ist
- Pipes funktionieren nur mit *stdout*! Man kann allerdings auch den Fehlerstrom in den Standardausgabestrom umleiten.
- Trick: *stdout* und *stderr* tauschen: *Befehl* 3>&1 1>&2 2>&3

Eine Beispiel-Pipeline (1)



Eine Pipeline könnte z. B. folgendermaßen⁷ aussehen:

Kommandozeile mit Pipes

```
grep "ion$" /korpora/Limas/limas.pp | \  
  sort | \  
  uniq -c | \  
  sort -rn | \  
  less 2>/dev/null
```

Übung

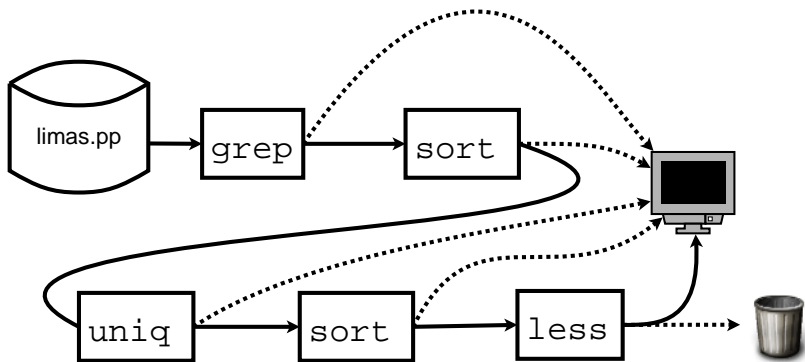
Probieren Sie die obige Zeile aus und überlegen Sie, wofür sie gut sein könnte. Woher bekommen jeweils die Befehle ihre Eingabe und wohin geben sie aus? Was passiert mit Fehlermeldungen?

⁷Ein Backslash am Ende der Zeile bedeutet: die Zeile geht hier weiter, also *nicht*  drücken und den Backslash nicht mit eingeben! 

Eine Beispiel-Pipeline (2)

```
grep "ion$" /korpora/Limas/limas.pp | sort |  
uniq -c | sort -rn | less 2>/dev/null
```

Die Kommandozeile von der vorigen Folie lässt sich folgendermaßen veranschaulichen:



Vorder- und Hintergrundprozesse

- Unix-Shells bieten die Möglichkeit, Programme im Hintergrund ausführen zu lassen, d. h. ohne das Terminal zu blockieren.
- Starten eines Hintergrundprozesses: Kommandozeile mit Kaufmannsund abschließen (*Befehl* [*Argumente*] &)
- **Vorsicht:** Ausgaben eines Hintergrundprozesses gehen weiterhin an das Terminal; wartet er auf Eingaben, hängt er, bis er wieder in den Vordergrund geholt wird!

Vorder- und Hintergrundprozesse

- Unix-Shells bieten die Möglichkeit, Programme im Hintergrund ausführen zu lassen, d. h. ohne das Terminal zu blockieren.
- Starten eines Hintergrundprozesses: Kommandozeile mit Kaufmannsund abschließen (*Befehl* [*Argumente*] &)
- **Vorsicht:** Ausgaben eines Hintergrundprozesses gehen weiterhin an das Terminal; wartet er auf Eingaben, hängt er, bis er wieder in den Vordergrund geholt wird!

Hintergrundprozesse

```
$ firefox&  
[1] 2773  
$
```

Ausgabe: Job-Nummer in eckigen Klammern; dahinter die Prozess-ID.

Job Control

Mittels *Job Control* kann in einem Terminal zwischen mehreren laufenden und/oder gestoppten Programmen hin- und hergeschaltet werden.

- Einen laufenden Prozess stoppen: `Strg` + `Z`
- Im Terminal laufende und gestoppte Prozesse anzeigen: `jobs`
- Einen gestoppten Prozess im Hintergrund weiterlaufen lassen:
`bg [jobnr.]`
- Einen gestoppten oder Hintergrundprozess wieder in den Vordergrund holen: `fg [jobnr.]`

Job Control

Mittels *Job Control* kann in einem Terminal zwischen mehreren laufenden und/oder gestoppten Programmen hin- und hergeschaltet werden.

- Einen laufenden Prozess stoppen: `Strg` + `Z`
- Im Terminal laufende und gestoppte Prozesse anzeigen: `jobs`
- Einen gestoppten Prozess im Hintergrund weiterlaufen lassen: `bg [jobnr.]`
- Einen gestoppten oder Hintergrundprozess wieder in den Vordergrund holen: `fg [jobnr.]`

Noch mehr Befehle:

- `ps` : Laufende Prozesse (mit PID, Terminal, Rechenzeit etc.) anzeigen
- `kill PID ...` : Prozess zwangsweise beenden (ganz brutal: `kill -9 PID ...`)
- `pidof Programmname` : PID eines laufenden Prozesses ermitteln

Shell-Variablen (1)

- Über sog. *Umgebungsvariablen* sind viele Aspekte einer Shell und anderer Programme konfigurierbar. Sie dienen außerdem der Zwischenspeicherung von Daten in Shellskripts.

Shell-Variablen (1)

- Über sog. *Umgebungsvariablen* sind viele Aspekte einer Shell und anderer Programme konfigurierbar. Sie dienen außerdem der Zwischenspeicherung von Daten in Shellskripts.
- *Name=Wert* weist der Variablen *Name* den *Wert* zu.
Vorsicht: kein Leerzeichen vor oder hinter dem Gleichheitszeichen!
- Damit diese Variable auch in aus dieser Shell(!) gestarteten Programmen bekannt ist, muss sie *exportiert* werden:
 - **export** *Name*
 - Kombiniert: **export** *Name=Wert*
- Üblicherweise werden Variablennamen groß geschrieben, dies ist aber nicht zwingend.

Shell-Variablen (1)

- Über sog. *Umgebungsvariablen* sind viele Aspekte einer Shell und anderer Programme konfigurierbar. Sie dienen außerdem der Zwischenspeicherung von Daten in Shellskripts.
- *Name=Wert* weist der Variablen *Name* den *Wert* zu.
Vorsicht: kein Leerzeichen vor oder hinter dem Gleichheitszeichen!
- Damit diese Variable auch in aus dieser Shell(!) gestarteten Programmen bekannt ist, muss sie *exportiert* werden:
 - **export** *Name*
 - Kombiniert: **export** *Name=Wert*
- Üblicherweise werden Variablennamen groß geschrieben, dies ist aber nicht zwingend.
- In Scripts sind diverse Variablen vorbelegt, z. B. \$1...\$9 mit einzelnen Argumenten, \$* mit sämtlichen Argumenten, \$0 mit dem Scriptnamen, etc.

Shell-Variablen (2)


- Um den Wert einer Variablen wieder zu benutzen, z. B. auf einer Kommandozeile, wird ihrem Namen ein Dollarzeichen vorangestellt.
- **set** zeigt alle momentan belegten Variablen und ihre Werte an.
- **unset** *Name* löscht eine Variable aus der Umgebung.

Variablenbenutzung

```
$ LSOPTS="--format=long --all"
$ ls $LSOPTS
drwx-----  2 msbethke users      4096 2007-01-18 21:03 Desktop
-rw-r--r--   1 msbethke users        399 2006-12-04 13:05 .bashrc
-rw-r--r--   1 msbethke users      1396 2006-11-17 15:29 .vimrc
$ set | grep LSOPTS
LSOPTS='--format=long --all'
$ unset LSOPTS
```

Wichtige Variablennamen

- PS1** Konfiguriert den primären Shell-Prompt (s. Bash-Manual für Ersetzungszeichen)
- PATH** Liste der Verzeichnisse, in denen die Shell nach Kommandos sucht; getrennt mit Doppelpunkten.
- LANG** Zu verwendende Sprache und Zeichensatz, z. B. `de_DE@euro` oder `ko_KR.utf8`.
Vollständige Liste mit `locale -a`
- HOME** Enthält den Namen des Home-Verzeichnisses
- USER** Der Benutzername, unter dem die Shell läuft
- EDITOR** Zu verwendender Editor für den Fall, dass ein Programm einen externen Editor aufrufen muss.
- HISTSIZE** Anzahl der Kommandos, die die *bash* in der „History“ behält.
- DISPLAY** Teilt X11-Programmen mit, auf welchem Display⁸ sie geöffnet werden sollen.

⁸Außer bei Mehr-Bildschirm-Anzeigen synonym mit „Monitor“ 

Befehlssubstitution

- Die Shell erlaubt es, Ausgaben eines Befehls (oder mehrerer) wie einen Variableninhalt auf der Kommandozeile einzufügen.
- Syntax: `'Befehl'` oder `$(Befehl)`

Befehlssubstitution

- Die Shell erlaubt es, Ausgaben eines Befehls (oder mehrerer) wie einen Variableninhalt auf der Kommandozeile einzufügen.
- Syntax: `'Befehl'` oder `$(Befehl)`

Reguläre Ausdrücke statt Wildcards

„Zeige mir detailliert alle Dateien, deren Name keine Ziffern enthält“

```
$ ls -l $(ls | egrep "^[^[:digit:]]+$")
```

```
test.dvi
```

```
test.tex
```

```
universal.tex
```

Editoren

- Programme zum Eingeben und Verändern von Texten

Editoren

- Programme zum Eingeben und Verändern von Texten
- Editor vs. Textverarbeitungsprogramm
 - Schriftattribute (fett, kursive, Größe, ...)
 - Druckfunktionen, Paginierung
 - WYSIWYG („*you see ASCII, you get ASCII*“)

Editoren

- Programme zum Eingeben und Verändern von Texten
- Editor vs. Textverarbeitungsprogramm
 - Schriftattribute (fett, kursive, Größe, ...)
 - Druckfunktionen, Paginierung
 - WYSIWYG („*you see ASCII, you get ASCII*“)
 - + Mächtige Textmanipulationskommandos
 - + Reguläre Ausdrücke, Makros
 - + Unterstützung für viele Programmiersprachen
 - + Erweiterbarkeit

GNU Emacs

- Der Standard-Editor der meisten Linux-Distributionen
- Verfügbar für quasi jedes Unix-Derivat sowie Windows (→ CygWin)
- Akronym: **E**ditor **MAC**ro**S**

GNU Emacs

- Der Standard-Editor der meisten Linux-Distributionen
- Verfügbar für quasi jedes Unix-Derivat sowie Windows (→ CygWin)
- Akronym: **E**ditor **MAC**ro**S**
- Ursprünglich als Makropaket seit 1976 entwickelt, seit Mitte der 80er als GNU-Projekt.
- Geschrieben in dem LISP-Dialekt ELISP, der auch dem Benutzer für Erweiterungen zur Verfügung steht.
- Zwei verschiedene Inkarnationen des ursprünglichen Emacs: *GNU Emacs* und *XEmacs* (Unterschiede sind nur bei über diesen Kurs hinausgehenden Erweiterungen relevant)

vi

- „Visual“-Modus des Zeileneditors *ex*⁹
- Entwickelt seit 1976, von Bell Labs als kommerzielles Programm in UNIX V aufgenommen.
- Optimiert für langsame Datenleitungen (Modem etc.)

⁹Traditionell – heutzutage ist *ex* ein eingeschränkter Modus des *vi*-Programms

vi

- „Visual“-Modus des Zeileneditors *ex*⁹
- Entwickelt seit 1976, von Bell Labs als kommerzielles Programm in UNIX V aufgenommen.
- Optimiert für langsame Datenleitungen (Modem etc.)
- Heutzutage viele Clones mit unterschiedlichen Erweiterungen: *nvi*, *elvis*, *vile* und v. a. *vim*
- Zum Teil deutliche Unterschiede bei über den Funktionsumfang des Original-*vi* hinausgehenden Kommandos
- *vim* ist zu empfehlen bzgl. Unterstützung für natürliche und Programmiersprachen sowie Kodierungen

⁹Traditionell – heutzutage ist *ex* ein eingeschränkter Modus des *vi*-Programms

Emacs

- Starten: aus der Shell mit `emacs [Datei] ...`

Emacs

- Starten: aus der Shell mit `emacs [Datei] ...`
- Nach dem Starten ist sofort Texteingabe möglich
- Befehle durch Drücken von `Strg` (abgekürzt „C-“ für „Control“) und/oder `Alt` (bzw. „M-“ für „Meta“) zusammen mit anderen Tasten.

Emacs

- Starten: aus der Shell mit `emacs [Datei] ...`
- Nach dem Starten ist sofort Texteingabe möglich
- Befehle durch Drücken von `Strg` (abgekürzt „C-“ für „Control“) und/oder `Alt` (bzw. „M-“ für „Meta“) zusammen mit anderen Tasten.

Grundlegende Befehle:

- `C-x C-f` Datei laden (Name wird abgefragt)
- `C-x C-s` Datei speichern (Name wird abgefragt, falls noch keiner vergeben wurde)
- `C-x C-w` Datei unter neuem Namen speichern
- `C-x C-i` Datei an Cursorposition einfügen
- `C-g` Angefangenes Kommando abbrechen (ggf. mehrfach drücken)

Cursorbewegung

Außer den Cursortasten gibt es auch schnellere Methoden, an eine Stelle im Text zu gelangen:

`C-a` Anfang der Zeile

`C-e` Ende der Zeile

`M-CursorLinks` Ein Wort zurück

`M-CursorRechts` Ein Wort vor

`M-<` Anfang des Textes

`M->` Ende des Textes

`Bild↑` Eine Seite zurück

`Bild↓` Eine Seite vor

`C-s` Text suchen

Die *Region* und Cut/Copy/Paste

- Viele Operationen in Emacs beziehen sich auf die *Region*, den Bereich zwischen der *Mark*[ierung] und der aktuellen Cursorposition.
- Markierung setzen mit C-SPC oder C-@
Operationen:
 - C-w** („wipe“, → cut) Löschen der Region und Übernehmen des gelöschten Teils in den *Ringbuffer*
 - M-w** (→ copy) Kopieren der Region in den Ringbuffer
 - C-y** („yank“, → paste) Den Ringbuffer an der Cursorposition einfügen

Buffers

- Ein geladener Text befindet sich als Kopie der Datei in einem *Buffer*
- Ein Buffer kann zu jeder Zeit einem Textfenster zugeordnet oder aber versteckt sein.
- Mit `C-x b` wird ein Textfenster auf einen anderen Buffer umgeschaltet (`C-x C-b` zeigt eine Liste).
- `C-x k` löscht den aktuellen Buffer.
- `C-x 2` bzw. `C-x 3` spalten das aktuelle Textfenster horizontal bzw. vertikal auf, so dass mehrere Buffer zugleich angezeigt werden können.
- `C-x o` („other“) wechselt zum nächsten Textfenster.

Suchen und Ersetzen

- C-s startet die *inkrementelle Suche* (s. Doku, bzw. ausprobieren, wie das funktioniert!)
- Einfaches suchen und ersetzen startet auf M-%
 - Suchbegriff und Ersetzungstext werden abgefragt
 - Treffer werden nacheinander angezeigt
 - Bei jedem Treffer kann wahlweise
 - y ersetzt werden
 - n nicht ersetzt werden
 - ! dieser und alle weiteren ersetzt werden
 - q die Suche abgebrochen werden
- Suchen und ersetzen von regulären Ausdrücken:
M-x query-replace-regex (ersetzt ein als regulären Ausdruck angegebenes Muster durch einen Text)

Makros

- Makros stehen an der Schnittstelle zwischen einfacher Bedienung des Editors und programmgesteuerter Automatisierung von Abläufen
- Sie erlauben es, Befehlssequenzen aufzuzeichnen und auf Kommando wieder beliebig oft abzuspielen.

Makros

- Makros stehen an der Schnittstelle zwischen einfacher Bedienung des Editors und programmgesteuerter Automatisierung von Abläufen
- Sie erlauben es, Befehlssequenzen aufzuzeichnen und auf Kommando wieder beliebig oft abzuspielen.
- Ein Emacs-Makro wird mit C-x (begonnen und mit C-x) beendet. Alle Kommandos dazwischen werden in das Makro übernommen und mit C-x e auf einmal ausgeführt.

Beispiel-Makro: LaTeX-Environment

```
C-x ( C-SPC M-← M-w \begin{ C-e } ← ←  
\end{ C-y C-e } ↑ C-x )
```

vi(m)

- Starten: aus der Shell mit `vi [-o] [Datei] ...`
- Im Unterschied zum Emacs werden nur mit der Option `-o` gleich alle als Argumente angegebenen Dateien angezeigt.
- `vi` läuft in der Shell; `gvim` öffnet ein eigenes Fenster.


¹⁰Außerdem: `a`, `o` und einige andere.

vi(m)

- Starten: aus der Shell mit `vi [-o] [Datei] ...`
- Im Unterschied zum Emacs werden nur mit der Option `-o` gleich alle als Argumente angegebenen Dateien angezeigt.
- `vi` läuft in der Shell; `gvim` öffnet ein eigenes Fenster.
- Nach dem Starten befindet sich der Editor im *Kommandomodus*
- Befehle werden im Kommandomodus als ein oder mehrere Buchstaben eingegeben, nur selten mit Strg.
- *ex*-Befehle beginnen mit Doppelpunkt und werden mit ← abgeschlossen.

¹⁰Außerdem: a, o und einige andere.

vi(m)

- Starten: aus der Shell mit `vi [-o] [Datei] ...`
- Im Unterschied zum Emacs werden nur mit der Option `-o` gleich alle als Argumente angegebenen Dateien angezeigt.
- `vi` läuft in der Shell; `gvim` öffnet ein eigenes Fenster.
- Nach dem Starten befindet sich der Editor im *Kommandomodus*
- Befehle werden im Kommandomodus als ein oder mehrere Buchstaben eingegeben, nur selten mit `Strg`.
- `ex`-Befehle beginnen mit Doppelpunkt und werden mit  abgeschlossen.
- `i` wechselt in den Eingabemodus¹⁰
- `Esc` wechselt in den Kommandomodus

¹⁰Außerdem: `a`, `o` und einige andere.

Grundlegende Befehle

- `:x [Datei]` Text speichern und Editor verlassen
- `:w [Datei]` Text speichern (unter neuem Namen, falls angegeben)
- `:e [Datei]` Angegebene Datei bearbeiten oder aktuelle neu laden
 - `:q` Editor verlassen¹¹ und nicht speichern
 - `:q!` Editor verlassen und nicht speichern, auch wenn ungespeicherte Änderungen vorhanden sind
 - `u` Letzte Änderung rückgängig machen (*undo*)
 - `.` Letztes Kommando wiederholen

Der Cursor kann in *vim* mit den Cursortasten bewegt werden; die ursprüngliche *vi*-Belegung funktioniert aber auch:

`h` Links

`l` Rechts

`k` Hoch

`j` Runter



¹¹Bzw. aktuelles Fenster schließen

Weitere Bewegungskommandos

- | | |
|--|---|
| w Zum Anfang des nächsten Wortes | H In die linke obere Ecke (<i>Home</i>) |
| W Zum Anfang des nächsten WORTES ¹² | M An den Anfang der mittleren Zeile |
| e Zum Ende des aktuellen Wortes | R In die linke untere Ecke |
| b Zum Anfang des vorigen Wortes (<i>back</i>) | gg Zum Anfang des Textes |
| ^ Zum ersten Zeichen in der Zeile | G Zum Ende des Textes |
| 0 An den Anfang der Zeile | nG In Zeile <i>n</i> |
| \$ Ans Ende der Zeile | fZ Auf das nächste Vorkommen des Zeichens <i>Z</i> |
| | tZ Vor das nächste Vorkommen des Zeichens <i>Z</i> |

¹²Stoppt nur bei Leerraum, nicht bei Satzzeichen. 

Moduswechsel und einfaches edieren

- i An der Cursorposition einfügen (*insert*)
- I Am Zeilenanfang einfügen
- a Nach der Cursorposition einfügen (*append*)
- A Am Zeilenende einfügen
- o Neue Zeile unter der aktuellen einfügen (*open*)
- O Neue Zeile über der aktuellen einfügen
- x Zeichen unter dem Cursor löschen (wie )
- X Zeichen links vom Cursor löschen (wie )
- s Zeichen unter dem Cursor löschen und in den Einfügemodus wechseln (*substitute*)
- cBew.* Text entsprechend *Bew*[egung] löschen und in den Einfügemodus wechseln (*change*)
- C Bis zum Zeilenende ändern
- R An Cursorposition in den Überschreibmodus wechseln (*replace*)

Grundlegendes cut/copy/paste

- dd** Zeile löschen
 - D** Von der Cursorposition bis zum Zeilenende löschen
- yy** Zeile kopieren
 - p** Kopiertes nach der Cursorposition einfügen (Zeilen: darunter)
 - P** Kopiertes vor der Cursorposition einfügen (Zeilen: darüber)
- v** „Visuelles“ Markieren starten. Danach:
 - d** Markierten Block löschen
 - y** Markierten Block kopieren
 - p** Markierten Block durch Kopierpuffer ersetzen
- V** Visuelles Markieren zeilenweise
- C-v** Visuelles Markieren blockweise

Zähler

Jedes *vi*-Kommando kann mit einem Zähler kombiniert und entsprechend oft ausgeführt werden. Zum Beispiel:

20j Cursor 20 Zeilen nach unten bewegen

5w Cursor fünf Wörter weiter bewegen

3fA Zum dritten Vorkommen von „A“ rechts vom Cursor

2yy Zwei Zeilen kopieren

10x 10 Zeichen rechts vom Cursor löschen

80iText *Text* (beendet mit Esc) 80-mal einfügen

3p Kopierte Zeile dreimal unter dem Cursor einfügen

Kombination von Edier- und Bewegungskommandos

Die meisten Edierkommandos akzeptieren einen Bereich, auf den sie angewendet werden sollen. Dieser Bereich kann durch ein Bewegungskommando (auch incl. Zähler) angegeben werden:

- `cw` Bis zum Wortende ändern (*change word*)
- `d5w` Fünf Wörter löschen (synonym: `5dw`; *delete 5 words*)
- `d5W` Fünf WÖRTER löschen (*delete 5 WORDS*)
- `y3j` Aktuelle Zeile und die drei darunter kopieren

Kombination von Edier- und Bewegungskommandos

Die meisten Edierkommandos akzeptieren einen Bereich, auf den sie angewendet werden sollen. Dieser Bereich kann durch ein Bewegungskommando (auch incl. Zähler) angegeben werden:

- cw** Bis zum Wortende ändern (*change word*)
- d5w** Fünf Wörter löschen (synonym: *5dw*; *delete 5 words*)
- d5W** Fünf WÖRTER löschen (*delete 5 WORDS*)
- y3j** Aktuelle Zeile und die drei darunter kopieren

Der Bereich kann auch ein sog. *Textobjekt* sein, bei dem dann die Cursorposition egal ist:

- 5caw** Fünf Wörter ändern (*a word*)
- y2ap** Zwei Absätze kopieren (*a paragraph*)
- dis** Einen Satz ohne das Satzzeichen löschen (*inner sentence*)
- ya(** Einen geklammerten Block kopieren (auch: *ab*, *a block*)

Suchen und Ersetzen

- / Nach einem Text oder Muster vorwärts suchen
- ? Nach einem Text oder Muster rückwärts suchen
- n Letzte Suche in der gleichen Richtung wiederholen (*next*)
- N Letzte Suche in der Gegenrichtung wiederholen

Suchen und Ersetzen

- / Nach einem Text oder Muster vorwärts suchen
- ? Nach einem Text oder Muster rückwärts suchen
- n Letzte Suche in der gleichen Richtung wiederholen (*next*)
- N Letzte Suche in der Gegenrichtung wiederholen

Ersetzen mit dem *ex*-Kommando **s///** (*substitute*):

: [Bereich] s/Muster / [Ersatz] / [Flags]

Zum Beispiel: „Schnauze“ durch „Mund“ ersetzen.

:%s/Schnauze/Mund/gc

Suchen und Ersetzen

- / Nach einem Text oder Muster vorwärts suchen
- ? Nach einem Text oder Muster rückwärts suchen
- n Letzte Suche in der gleichen Richtung wiederholen (*next*)
- N Letzte Suche in der Gegenrichtung wiederholen

Ersetzen mit dem *ex*-Kommando **:s///** (*substitute*):

: [Bereich] s/Muster / [Ersatz] / [Flags]

Zum Beispiel: „Schnauze“ durch „Mund“ ersetzen.

:%s/Schnauze/Mund/gc

Wichtige Flags:

- g Jedes Vorkommen ersetzen, nicht nur das jeweils erste auf der Zeile (*global*)
- i Groß-/Kleinschreibung ignorieren (*ignore case*)
- c Jede Ersetzung bestätigen lassen (*confirm*)

ex-Bereiche

Der Zeileneditor `ex` kann seine Kommandos wahlweise auf verschiedene Textbereiche anwenden.

ex-Bereiche

Der Zeileneditor `ex` kann seine Kommandos wahlweise auf verschiedene Textbereiche anwenden.

Allgemeines Format: `:Anfang,Ende Kommando`

bzw.: `:Anfang;Relative_Pos. Kommando`

ex-Bereiche

Der Zeileneditor *ex* kann seine Kommandos wahlweise auf verschiedene Textbereiche anwenden.

Allgemeines Format: *:Anfang,Ende Kommando*

bzw.: *:Anfang;Relative_Pos. Kommando*

Anfang und *Ende* können Zeilennummern sein, oder aber Symbole:

- . Aktuelle Zeile
- +*n* *n* Zeilen nach der aktuellen Zeile
- n* *n* Zeilen vor der aktuellen Zeile
- 0 Anfang des Textes
- \$ Ende des Textes
- /*Muster*/ Nächste Zeile, die *Muster* enthält
- % Abkürzung für 0,\$: kompletter Text

ex-Bereiche: Beispiele

```
:.,$s/Worscht/Wurst/
```

Von der aktuellen Zeile bis zum Dateiende jedes erste Vorkommen von „Worscht“ auf der Zeile durch „Wurst“ ersetzen.

```
:0,+5w neu.txt
```

Den Text vom Dateianfang bis fünf Zeilen unter der aktuellen Zeile in die Datei `neu.txt` schreiben.

```
:/hallo/;-5w hallo.txt
```

Die nächste Zeile, die „hallo“ enthält, sowie die fünf Zeilen davor in die Datei `hallo.txt` schreiben (**Achtung**: Semikolon statt Komma ändert die Semantik des „-5“ nach „relativ zur ersten Positionsangabe, nicht zur aktuellen Zeile“!).

Mehrere Dateien editieren

Mittels *ex*-Kommandos kann das *vim*-Fenster geteilt werden:

`:sp` Fenster horizontal teilen

`:sp Datei` In einem neuem Fenster *Datei* öffnen.

`:vert sp` Fenster vertikal teilen (opt. *Datei* wie oben)

`C-w` `[Strg]` + `[w]` leitet ein Fenster-Kommando ein:

`C-w` Cursor ins nächste Fenster

`C-p` Cursor ins vorige Fenster (*previous*)

`+/-` Fenster um eine Zeile höher/niedriger machen

`>/<` Fenster um eine Spalte breiter/schmäler machen

`_` Fenster maximieren

`=` Alle Fenster gleich hoch/breit machen

`o` Alle Fenster außer dem aktuellen schließen (*only*)

Makros

- Makroaufzeichnungen starten: im Kommandomodus mit **qB**
 - *B* kann ein beliebiger Buchstabe oder Ziffer sein.
 - Bei groß geschriebenem *B* wird das unter *B* bereits gespeicherte Makro erweitert.
- Beenden der Aufzeichnung mit **q**
- Abspielen des Makros mit **@B**

Beispiel-Makro: LaTeX-Environment

```

q e b y w i \begin{   [Esc] A }  ← ← \end{   [Esc]
p a }   ↑   [Esc] q

```

„Aufzeichnung in Register e; *back*; *yank-word*; *insert*: \begin{; Kommandomodus; *append-at-end*: Leerzeile \end{; *put*; *append*: }; *Cursor-hoch*; Aufzeichnung Ende“

Weitere Kommandos

- `:help [Thema]` Hilfe aufrufen, optional zu einem Thema
 - `C-]` Auf der Hilfe-Seite zu dem Stichwort springen, auf dem sich der Cursor befindet¹³
 - `C-w C-]` Hilfe zum Stichwort unter dem Cursor in neuen Fenster öffnen
 - `C-t` Zum letzten Hilfethema zurückspringen
 - `~` Zeichen unter dem Cursor von Groß- nach Kleinschreibung und umgekehrt ändern
 - `xp` Zeichen unter dem Cursor mit dem dahinter tauschen (*transpose*, eigentlich nur ein „Idiom“)
 - `mZ` Markierung *Z* setzen (*Z*: bel. Buchstabe/Ziffer)
 - `'Z` Zu Markierung *Z* springen

¹³Allgemeiner Mechanismus, funktioniert auch in vielen Programmiersprachen-Modi.

Befehle zur Textbearbeitung (1)

Die folgenden Befehle nehmen alle optionale Datei-Argumente; fehlen diese, wird wie üblich `stdin` bearbeitet.

`grep Muster` Sucht zeilenweise nach *Muster* und gibt gefundene Zeilen aus.

`tac` Kehrt die Zeilenreihenfolge um („cat rückwärts“).

`cut` Gibt ausgewählte Spalten aus.

`paste` Fügt Dateien¹⁴ spaltenweise zusammen, getrennt mit (per default) Tabulatorzeichen.

`sort` Sortiert die Eingabe zeilenweise, wahlweise alphabetisch oder numerisch.

`uniq` Gibt aufeinanderfolgende gleiche Zeilen nur je einmal aus (d. h. bei sortierter Eingabe die Menge der unterschiedlichen Zeilen)

¹⁴Funktioniert auch mit nur einer Eingabe von `stdin`, nur ist das nicht besonders sinnvoll!

Befehle zur Textbearbeitung (2)

Hier gilt natürlich das selbe wie für die Befehle der vorigen Folie.


wc Zählt Zeichen, Wörter und Zeilen (*word-count*)

head Gibt eine bestimmte Anzahl Zeilen (Option `-n`) vom Anfang der Eingabe aus. Auch mit negativem Argument: alle *außer* n Zeilen am Ende (z. B. **head -n-20**).

tail Gibt eine bestimmte Anzahl Zeilen vom Ende der Eingabe aus. Auch: alle *außer* n Zeilen am Anfang mit explizit positivem Argument (z. B. **tail -n+20**).

fold Umbricht lange Eingabezeilen auf eine bestimmte maximale Breite.

sed „Stream Editor“, stellt eine einfache¹⁵ Programmiersprache zur Textmanipulation zur Verfügung, grob vergleichbar mit dem in *vi* eingebauten *ex*.

¹⁵Im Sinne von „wenig mächtig“, nicht „einfach zu lesen“ 

tar

`tar` („tape archiver“) dient heutzutage seltener der Archivierung auf Band als vielmehr dem Zusammenpacken mehrerer Dateien in eine Archivdatei, die dann z. B. komprimiert werden kann.

- Aufruf: `tar [Option...] Kommando [Datei...]`
- Kommandos können mit und ohne führenden Bindestrich angegeben werden

tar

tar („tape archiver“) dient heutzutage seltener der Archivierung auf Band als vielmehr dem Zusammenpacken mehrerer Dateien in eine Archivdatei, die dann z. B. komprimiert werden kann.

- Aufruf: **tar** [*Option...*] *Kommando* [*Datei...*]
- Kommandos können mit und ohne führenden Bindestrich angegeben werden

Die wichtigsten Kommandos:

- c** Archiv anlegen (*create*)
- t** Archivinhalt auflisten (*list*)
- x** Archiv auspacken (*extract*)

Übliche Optionen:

- f** *File* Diese Archivdatei bearbeiten

- v** Fortschritt anzeigen (*verbose*, auch **-vv**, *very verbose*)
- z** Archiv mit gzip (de)komprimieren
- j** Archiv mit bzip2 (de)komprimieren

tar-Idioms

tar klingt komplizierter als es ist; meist reicht es, sich einige häufige Idioms zu merken:

```
tar -cvf archiv.tar datei1 datei2
```

Unkomprimiertes Archiv `archiv.tar` anlegen und `datei1` und `datei2` hineinpacken; dabei die Namen der Dateien anzeigen, während diese archiviert werden.

```
tar -xzvfvf archiv.tar.gz
```

gzip-komprimiertes Archiv `archiv.tar.gz` ins aktuelle Verzeichnis auspacken; dabei die Dateien ausführlich anzeigen.


```
tar -jxf archiv.tar.bz2 datei1
```

`datei1` aus dem bzip2-komprimierten Archiv `archiv.tar.bz2` ins aktuelle Verzeichnis auspacken.

Reguläre Ausdrücke

- Reguläre Ausdrücke (*Regular Expressions*, oft abgekürzt als *RegEx*) sind Zeichenketten, die Mengen von anderen Zeichenketten beschreiben.
- Sie sind damit Grammatiken für eine einfache Sprache¹⁶.

¹⁶Genaugenommen: eine *reguläre Sprache* bzw. Typ 3 der Chomsky-Hierarchie, vgl. Hausser 2000, Abschn. 8.1.1f.

¹⁷Entweder als Sprachbestandteil oder externe Bibliothek 

Reguläre Ausdrücke

- Reguläre Ausdrücke (*Regular Expressions*, oft abgekürzt als *RegEx*) sind Zeichenketten, die Mengen von anderen Zeichenketten beschreiben.
- Sie sind damit Grammatiken für eine einfache Sprache¹⁶.
- Meist werden sie zum Suchen von Mustern in Texten eingesetzt.
 - Jeder ernstzunehmende Editor sowie viele Kommandozeilenwerkzeuge und die allermeisten Programmiersprachen¹⁷ bieten entsprechende Funktionalität.


¹⁶Genaugenommen: eine *reguläre Sprache* bzw. Typ 3 der Chomsky-Hierarchie, vgl. Hausser 2000, Abschn. 8.1.1f.

¹⁷Entweder als Sprachbestandteil oder externe Bibliothek 

Reguläre Ausdrücke

- Reguläre Ausdrücke (*Regular Expressions*, oft abgekürzt als *Regex*) sind Zeichenketten, die Mengen von anderen Zeichenketten beschreiben.
- Sie sind damit Grammatiken für eine einfache Sprache¹⁶.
- Meist werden sie zum Suchen von Mustern in Texten eingesetzt.
- Jeder ernstzunehmende Editor sowie viele Kommandozeilenwerkzeuge und die allermeisten Programmiersprachen¹⁷ bieten entsprechende Funktionalität.
- **Vorsicht:** nicht alle Werkzeuge verwenden die gleiche Syntax! Manche verstehen auch nur bestimmte *Regex*-Elemente.

¹⁶Genaugenommen: eine *reguläre Sprache* bzw. Typ 3 der Chomsky-Hierarchie, vgl. Hausser 2000, Abschn. 8.1.1f.

¹⁷Entweder als Sprachbestandteil oder externe Bibliothek 

Reguläre Ausdrücke: Werkzeuge

grep Durchsucht Dateien oder Datenströme nach einfachen regulären Ausdrücken (Zeichenklassen incl. POSIX-Klassen, *, ^, \$). Wichtige Option zum testen: `--color`

egrep¹⁸ Wie `grep`, zusätzlich Gruppierung und Alternation, +, ?, \>, \< und numerische Quantoren.

vim Unterstützt alles in `egrep` und vieles mehr, manchmal mit etwas unterschiedlicher Syntax¹⁹. Beschreibung mit **`:help pattern`**

emacs Wie in `egrep`, mit einer ganzen Reihe praktischer Erweiterungen; z. T. eigene Syntax wie in `vim`, z. B. Gruppierung mit `\(...\)`, aber weniger konfigurierbar.

¹⁸Genauso funktioniert `grep -e`; in Wirklichkeit sind beide ein und das selbe Programm.

¹⁹Konfigurierbar, vgl. **`:help magic`**

Reguläre Ausdrücke formal

Eine allgemeine Syntax für reguläre Ausdrücke

- 1 \emptyset ist RA
- 2 ϵ ist RA
- 3 $\forall a_i \in \Sigma : \underline{a_i}$ ist RA
- 4 x, y sind RA \implies
 - 1 $x \cup y$ ist RA (*Vereinigung*)
 - 2 xy ist RA (*Konkatenation*)
 - 3 x^* ist RA (*Kleene-Operator*)

Vorsicht: obiges ist die *Syntax der Syntaxbeschreibungssprache*, die ihrerseits die Syntax anderer Ausdrücke beschreibt!

Reguläre Ausdrücke: Literale und Zeichenklassen

- Buchstaben, Ziffern und viele Sonderzeichen stehen in RA für sich selbst; der RA „a“ steht also einfach für ein „a“.
- Das selbe gilt für Aneinanderreihungen derselben; der RA „hallo“ steht also für eben dieses Wort.

Reguläre Ausdrücke: Literale und Zeichenklassen

- Buchstaben, Ziffern und viele Sonderzeichen stehen in RA für sich selbst; der RA „a“ steht also einfach für ein „a“.
- Das selbe gilt für Aneinanderreihungen derselben; der RA „hallo“ steht also für eben dieses Wort.
- Eine *Zeichenklasse* in eckigen Klammern steht für *ein* beliebiges in der Klasse enthaltenes Zeichen; der RA „[Abc]“ steht also für ein großes „A“ oder ein kleines „b“ oder „c“.
- Zeichenklassen können Zeichen explizit auflisten oder *Bereiche* definieren: „[A-Z0-9]“ steht z. B. für alle Großbuchstaben und Ziffern.
- Eine eckige Klammer kann als erstes Element in eine Zeichenklasse aufgenommen werden, ein Bindestrich als erstes oder letztes. Zum Beispiel: [] a-z-]
- Ist das erste Zeichen der Klasse ein Zirkumflex oder *Caret*, wird die Klasse negiert und steht für irgendein *nicht* enthaltenes Zeichen. Zum Beispiel: [^aeiou]

Reguläre Ausdrücke: Spezialklassen

- Der Punkt steht für irgendein beliebiges Zeichen
- Der POSIX-Standard vergibt für häufig gebrauchte Zeichenklassen symbolische Namen, die wie ein Zeichen in einer Klasse gebraucht werden können. Die wichtigsten:
 - `[lower:]` Kleinbuchstaben – was als Buchstabe gilt, ist locale-abhängig!
 - `[upper:]` Großbuchstaben, dito
 - `[alpha:]` Buchstaben: `[lower:]` und `[upper:]`
 - `[digit:]` Ziffern: 0, 1, 2,... bis 9.
 - `[alnum:]` Alphanumerische Zeichen: `[alpha:]` und `[digit:]`
 - `[blank:]` Leerzeichen und Tabulator.
 - `[space:]` Whitespace: `[blank:]` sowie vertikaler Tabulator, Zeilen- und Seitenvorschub, Wagenrücklauf.
 - `[punct:]` Interpunktionszeichen: `,`, `-`, `!`, `"`, `#`, `$`, usw.

Reguläre Ausdrücke: Quantoren (1)

Quantoren geben an, wie oft das vorangehende *Atom*²⁰ in einem Ausdruck erscheinen darf.

- Der Stern oder *Kleene-Operator* bedeutet dabei „beliebig oft“, d. h. auch null-mal. So passt der Ausdruck „ab.*“ auf beliebig lange Zeichenketten, die mit „ab“ anfangen, auch „ab“ selbst.

²⁰Ein Atom ist entweder ein einzelnes Zeichen, eine Zeichenklasse oder ein Klammerausdruck.

Reguläre Ausdrücke: Quantoren (1)

Quantoren geben an, wie oft das vorangehende *Atom*²⁰ in einem Ausdruck erscheinen darf.

- Der Stern oder *Kleene-Operator* bedeutet dabei „beliebig oft“, d. h. auch null-mal. So passt der Ausdruck „ab.*“ auf beliebig lange Zeichenketten, die mit „ab“ anfangen, auch „ab“ selbst.
- Das Pluszeichen steht für „beliebig oft, aber mindestens einmal“. Die Ausdrücke „a+“ und „aa*“ sind also äquivalent.

²⁰Ein Atom ist entweder ein einzelnes Zeichen, eine Zeichenklasse oder ein Klammerausdruck.

Reguläre Ausdrücke: Quantoren (1)

Quantoren geben an, wie oft das vorangehende *Atom*²⁰ in einem Ausdruck erscheinen darf.

- Der Stern oder *Kleene-Operator* bedeutet dabei „beliebig oft“, d. h. auch null-mal. So passt der Ausdruck „ab.*“ auf beliebig lange Zeichenketten, die mit „ab“ anfangen, auch „ab“ selbst.
- Das Pluszeichen steht für „beliebig oft, aber mindestens einmal“. Die Ausdrücke „a+“ und „aa*“ sind also äquivalent.
- Das Fragezeichen bedeutet „optional“ bzw. „null- oder einmal“.

²⁰Ein Atom ist entweder ein einzelnes Zeichen, eine Zeichenklasse oder ein Klammerausdruck.

Reguläre Ausdrücke: Quantoren (1)

Quantoren geben an, wie oft das vorangehende *Atom*²⁰ in einem Ausdruck erscheinen darf.

- Der Stern oder *Kleene-Operator* bedeutet dabei „beliebig oft“, d. h. auch null-mal. So passt der Ausdruck „ab.*“ auf beliebig lange Zeichenketten, die mit „ab“ anfangen, auch „ab“ selbst.
- Das Pluszeichen steht für „beliebig oft, aber mindestens einmal“. Die Ausdrücke „a+“ und „aa*“ sind also äquivalent.
- Das Fragezeichen bedeutet „optional“ bzw. „null- oder einmal“.
- **Vorsicht:** der Ausdruck vor dem Quantor wird bei jedem erneuten Abpassen neu ausgewertet!

²⁰Ein Atom ist entweder ein einzelnes Zeichen, eine Zeichenklasse oder ein Klammerausdruck.

Reguläre Ausdrücke: Quantoren (1)

Quantoren geben an, wie oft das vorangehende *Atom*²⁰ in einem Ausdruck erscheinen darf.

- Der Stern oder *Kleene-Operator* bedeutet dabei „beliebig oft“, d. h. auch null-mal. So passt der Ausdruck „ab.*“ auf beliebig lange Zeichenketten, die mit „ab“ anfangen, auch „ab“ selbst.
- Das Pluszeichen steht für „beliebig oft, aber mindestens einmal“. Die Ausdrücke „a+“ und „aa*“ sind also äquivalent.
- Das Fragezeichen bedeutet „optional“ bzw. „null- oder einmal“.
- **Vorsicht:** der Ausdruck vor dem Quantor wird bei jedem erneuten Abpassen neu ausgewertet!
- *Emacs* implementiert alle der obigen Quantoren; *vim* benötigt einen vorangestellten Backslash für + und ?.

²⁰Ein Atom ist entweder ein einzelnes Zeichen, eine Zeichenklasse oder ein Klammerausdruck.

Reguläre Ausdrücke: Quantoren (2)

- Numerische Quantoren bestehen aus einer Zahl oder einem mit Komma getrennten Zahlenpaar in geschweiften Klammern.
 - Eine einzelne Zahl steht für „genau so oft“, z. B. „ $a\{5\}$ “ für „genau 5 aufeinanderfolgende a“.
 - Ein Zahlenpaar gibt die minimale und maximale Anzahl an, z. B. passt „ $S[ea]\{2,3\}.+$ “ auf „Seeadler“ und „Seattle“, nicht aber „Sessel“.
 - Eine der Minimum- bzw. Maximum-Angaben kann jeweils entfallen²¹ wobei dann null respektive unendlich angenommen wird:
 - ab $\{3,\}$: ein a gefolgt von mindestens 3 b
 - b. $\{,10\}$: ein b gefolgt von höchstens 10 beliebigen Zeichen

²¹Bei *egrep* nur das Maximum, d.h. $\{2,\}$ funktioniert, $\{,2\}$ aber nicht!

Reguläre Ausdrücke: Quantoren (2)

- Numerische Quantoren bestehen aus einer Zahl oder einem mit Komma getrennten Zahlenpaar in geschweiften Klammern.
 - Eine einzelne Zahl steht für „genau so oft“, z. B. „a{5}“ für „genau 5 aufeinanderfolgende a“.
 - Ein Zahlenpaar gibt die minimale und maximale Anzahl an, z. B. passt „S[ea]{2,3}.+“ auf „Seeadler“ und „Seattle“, nicht aber „Sessel“.
 - Eine der Minimum- bzw. Maximum-Angaben kann jeweils entfallen²¹ wobei dann null respektive unendlich angenommen wird:
 - ab{3,}: ein a gefolgt von mindestens 3 b
 - b.{,10}: ein b gefolgt von höchstens 10 beliebigen Zeichen
- *Emacs* unterstützt keine numerischen Quantoren; *vims* Syntax verlangt einen vorangestellten Backslash, z. B. ab\`{2,3}`

²¹Bei *egrep* nur das Maximum, d.h. `{2,}` funktioniert, `{,2}` aber nicht!

Reguläre Ausdrücke: Gruppierung und Alternation

- Mit runden Klammern können beliebig lange Teile eines RA zu *Atomen* zusammengefasst werden.

²²Wo dies nicht erwünscht ist, lässt es sich manchmal abschalten, z. B. in *PerlREs* mit `(?:...)`, in *vim* mit `\%(...)`.

Reguläre Ausdrücke: Gruppierung und Alternation

- Mit runden Klammern können beliebig lange Teile eines RA zu *Atomen* zusammengefasst werden.
 - Erlaubt die Anwendung von Quantoren auf den gesamten geklammerten Ausdruck.
 - Stellt i. d. R. den Text, auf den der Klammersausdruck passt, in einem speziellen Register zur Verfügung²², wo er z. B. in Ersetzungstexten benutzt werden kann.
- Beispiel: „(hallo){,2}“ passt auf die leere Zeichenkette(!), „hallo“ und „hallohallo“.

²²Wo dies nicht erwünscht ist, lässt es sich manchmal abschalten, z. B. in *PerlREs* mit `(?:...)`, in *vim* mit `\%(...)`.

Reguläre Ausdrücke: Gruppierung und Alternation

- Mit runden Klammern können beliebig lange Teile eines RA zu *Atomen* zusammengefasst werden.
 - Erlaubt die Anwendung von Quantoren auf den gesamten geklammerten Ausdruck.
 - Stellt i. d. R. den Text, auf den der Klammersausdruck passt, in einem speziellen Register zur Verfügung²², wo er z. B. in Ersetzungstexten benutzt werden kann.
- Beispiel: „(hallo){,2}“ passt auf die leere Zeichenkette(!), „hallo“ und „hallohallo“.
- Der senkrechte Strich (→„Pipe-Symbol“) markiert eine Alternation und wird üblicherweise mit Klammern benutzt, z. B. x(ABC|abc)y für „xABCy“ oder „xabcy“, nicht aber „xAbcy“ (vgl. „x[AaBbCc]{3}y“!).

²²Wo dies nicht erwünscht ist, lässt es sich manchmal abschalten, z. B. in *PerlREs* mit `(?:...)`, in *vim* mit `\%(...)`.

Reguläre Ausdrücke: Gruppierung und Alternation

- Mit runden Klammern können beliebig lange Teile eines RA zu *Atomen* zusammengefasst werden.
 - Erlaubt die Anwendung von Quantoren auf den gesamten geklammerten Ausdruck.
 - Stellt i. d. R. den Text, auf den der Klammersausdruck passt, in einem speziellen Register zur Verfügung²², wo er z. B. in Ersetzungstexten benutzt werden kann.
- Beispiel: „(hallo){,2}“ passt auf die leere Zeichenkette(!), „hallo“ und „hallohallo“.
- Der senkrechte Strich (→„Pipe-Symbol“) markiert eine Alternation und wird üblicherweise mit Klammern benutzt, z. B. x(ABC|abc)y für „xABCy“ oder „xabcy“, nicht aber „xAbcy“ (vgl. „x[AaBbCc]{3}y“!).
- Sowohl *Emacs* als auch *vim* (aber nicht *egrep*) brauchen Klammern und Alternationsstriche mit Backslash!

²²Wo dies nicht erwünscht ist, lässt es sich manchmal abschalten, z. B. in *PerlREs* mit `(?:...)`, in *vim* mit `\%(...)`.

Gierige Ausdrücke (1)

- Reguläre Ausdrücke sind normalerweise *gierig* („*greedy matching*“), d. h. sie versuchen, so viel Text wie möglich zu matchen.
- Ausprobieren mit `egrep --color` auf `limas.pp` und `limas.txt`: alle Wörter finden, die mit Großbuchstaben anfangen und auf „a“ enden.

Gierige Ausdrücke (1)

- Reguläre Ausdrücke sind normalerweise *gierig* („*greedy matching*“), d. h. sie versuchen, so viel Text wie möglich zu matchen.
- Ausprobieren mit `egrep --color` auf `limas.pp` und `limas.txt`: alle Wörter finden, die mit Großbuchstaben anfangen und auf „a“ enden.
 - Erster Versuch: `[[[:upper:]]]*a`
Ergebnis?
- Manche RA-Engines (Perl, Emacs, nicht `egrep` und `vim`) erlauben es, einen Quantor durch ein nachgestelltes Fragezeichen auf „nicht-gierig“ umzustellen, z. B. „`[[[:upper:]]]*?a`“: Von einem Großbuchstaben bis zum nächsten folgenden „a“ (Vorsicht, löst immer noch nicht ganz die obige Aufgabe!)

Gierige Ausdrücke (2)

- Lösung für `egrep`: Charakterisierung der erlaubten Zeichen statt des Punkt-Platzhalters.

Gierige Ausdrücke (2)

- Lösung für egrep: Charakterisierung der erlaubten Zeichen statt des Punkt-Platzhalters.
- Beispiel: „`[[:upper:]] [^]*a`“

Gierige Ausdrücke (2)

- Lösung für `egrep`: Charakterisierung der erlaubten Zeichen statt des Punkt-Platzhalters.
 - Beispiel: „`[[:upper:]] [^]*a`“
- Großbuchstabe gefolgt von beliebig vielen Nicht-Leerzeichen, gefolgt von einem `a` und einem Leerzeichen²³

²³Weiterer Sonderfall: Zeilenende! Benötigt weitere Spezialzeichen. 

Gierige Ausdrücke (2)

- Lösung für `egrep`: Charakterisierung der erlaubten Zeichen statt des Punkt-Platzhalters.
 - Beispiel: „`[[:upper:]] [^]*a`“
- Großbuchstabe gefolgt von beliebig vielen Nicht-Leerzeichen, gefolgt von einem `a` und einem Leerzeichen²³
- Lösung in `vim`: die Rolle des Fragezeichens wird von Minuszeichen in numerischen Quantoren übernommen (vgl. **:help {}**)
 - `\{-}` entspricht `*?`
 - `\{-,1}` entspricht `??`
 - `\{-1,}` entspricht `+?`

²³Weiterer Sonderfall: Zeilenende! Benötigt weitere Spezialzeichen. 

Weitere Sonderzeichen

Einige Muster repräsentieren abstrakte Einheiten im Text wie z. B. Wortgrenzen. Für diese existieren Sonderzeichen oder sog. *Escape-Sequenzen*, d. h. Kombinationen eines Backslash mit einem anderen Zeichen.

- ^ Zeilenanfang
 - \$ Zeilenende
 - \b Wortzwischenraum (Anfang oder Ende, auch bei Satzzeichen)
 - \B Alles außer Wortzwischenräumen
 - \< Zwischenraum am Wortanfang
 - \> Zwischenraum am Wortende
-
- \w *Word character*, entspricht `[[:alnum:]]`
 - \W *Non-word character*, entspricht `[^[:alnum:]]`

Übungen und Beispiele

Worauf passen diese Ausdrücke?

```
[[:lower:]].*?\b  
\<(ab|an|um)ge[^\k]+
```

Schreiben Sie reguläre Ausdrücke, die...

- 1 Datumsangaben in numerischer Form erkennen (Abwägen: Erkennung vs. Validierung!).
Test: *Datum.txt* in meinem Home-Verzeichnis.
- 2 Datumsangaben in numerischer Form und mit Monatsnamen erkennen.
- 3 Verbformen der 2. Pers. Singular Präsens Indikativ erkennen.

Ersetzung in *vim*: Worte tauschen

```
s/\(homen\|mulher\) \(\grande\|pequen[oa]\)/\2 \1/g
```

Übungen und Beispiele

Worauf passen diese Ausdrücke?

```
[[:lower:]].*?\b  
\<(ab|an|um)ge[^\k]+
```

Schreiben Sie reguläre Ausdrücke, die...


- 1 Datumsangaben in numerischer Form erkennen (Abwägen: Erkennung vs. Validierung!).
Test: *Datum.txt* in meinem Home-Verzeichnis.
- 2 Datumsangaben in numerischer Form und mit Monatsnamen erkennen.
- 3 Verbformen der 2. Pers. Singular Präsens Indikativ erkennen.

Ersetzung in *vim*: Worte tauschen

```
s/\(homen\|mulher\) \(\grande\|pequen[oa]\)/\2 \1/g
```


T_EX und L_AT_EX

- T_EX ist eine Makrosprache zum Textsatz.
- Entwickelt Ende der 70er von Donald Knuth für *The Art of Computer Programming*
- Erlaubt die weitgehend automatische Herstellung schöner Druckerzeugnisse, insbesondere solcher mit vielen mathematischen Formeln, auf beinahe beliebigen Systemen.

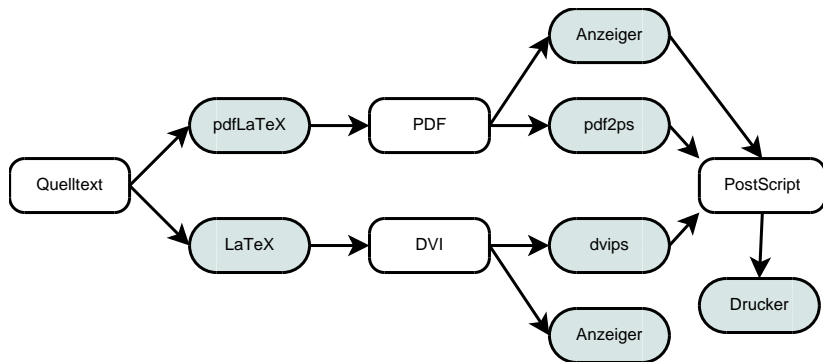
²⁴Entwickelt von Leslie Lamport; → „Lamport's T_EX“ 

T_EX und L_AT_EX

- T_EX ist eine Makrosprache zum Textsatz.
- Entwickelt Ende der 70er von Donald Knuth für *The Art of Computer Programming*
- Erlaubt die weitgehend automatische Herstellung schöner Druckerzeugnisse, insbesondere solcher mit vielen mathematischen Formeln, auf beinahe beliebigen Systemen.
- Reines T_EX ist relativ kompliziert und „technisch“ zu programmieren, deshalb gibt es verschiedene *Makropakete* für generisches Markup.
- Die bekanntesten sind L_AT_EX²⁴ und ConT_EXt; wir verwenden ersteres.
- L_AT_EX ist ein Quasi-Standard zum erstellen von wissenschaftlichen Publikationen.

²⁴Entwickelt von Leslie Lamport; → „Lamport's T_EX“ 

Die Verarbeitungskette von T_EX



Ein kleines L^AT_EX-Dokument

```
1  \documentclass[a4paper]{article}
2  \usepackage[ngerman]{babel}
3  \usepackage[latin1]{inputenc}
4  \begin{document}
5  \section{Erster Abschnitt}
6  Dies ist ein Textabsatz
7  und das hier gehört auch dazu.
8
9  Das ist noch einer.
10 \end{document}
```

Ein kleines L^AT_EX-Dokument

```
1  \documentclass[a4paper]{article}
2  \usepackage[ngerman]{babel}
3  \usepackage[latin1]{inputenc}
4  \begin{document}
5  \section{Erster Abschnitt}
6  Dies ist ein Textabsatz
7  und das hier gehört auch dazu.
8
9  Das ist noch einer.
10 \end{document}
```

- *Befehle* beginnen mit Backslash (\)
- *Argumente* erscheinen in eckigen oder geschweiften Klammern

Verarbeitung von L_AT_EX-Dokumenten

Aufruf des T_EX-Interpreters

```
$ latex mydoc.tex
This is pdfTeX, Version 3.141592-1.30.5-2.2 (Web2C 7.5.5)
entering extended mode
(./mydoc.tex
LaTeX2e <2003/12/01>
Babel <v3.8d> and hyphenation patterns [...]
(/usr/share/texmf/tex/latex/base/article.cls
[...])
Output written on mydoc.dvi (1 page, 392 bytes).
ranscript written on mydoc.log.
```

- Der Befehl zum Setzen eines L_AT_EX-Dokuments lautet **latex** oder **pdflatex**.
- Während **latex** das traditionelle DVI²⁵-Dateiformat erzeugt, produziert **pdflatex** direkt PDF²⁶.

²⁵DeVice Independent

²⁶Adobe Portable Document Format

Anzeige des Resultats

Das Resultat als DVI/PDF

1 Erster Absatz

Dies ist ein Textabsatz und das hier gehört auch dazu.
Das ist noch einer.

- PDF-Dokumente sind mit jedem beliebigen PDF-Anzeiger darzustellen: *xpdf*, *acroread*, *evince*, das PDF-Modul von *konqueror*, etc.
- Zur Anzeige von DVI-Dateien dient das Programm *xdvi* oder sein KDE-Äquivalent *kdvi* (z. B. via *konqueror*).

Dateitypen in T_EX

T_EX produziert beim Setzen eine ganze Reihe verschiedener Dateien, die sich im selben Verzeichnis wie der Quellcode ansammeln. Hier sind die wichtigsten:

- `.tex` T_EX-Quellcode, i. d. R. selbst erstellt.
- `.dvi` Traditionelles Ausgabeformat (*DeVice Independent*)
- `.pdf` Ausgabedatei von `pdftex` (*Portable Document Format*)
- `.log` Detailliertes Ablaufprotokoll des letzten T_EX-Aufrufs.
- `.toc` Inhaltsverzeichnis (*Table Of Contents*)
- `.lof` Abbildungsverzeichnis (*List Of Figures*)
- `.lot` Tabellenverzeichnis (*List Of Tables*)
- `.aux` Hilfsdatei für Querverweise etc.

Ein kleines LATEX-Dokument: Analyse

```
1 \documentclass[a4paper]{article}
```

- `\documentclass`: wir schreiben einen Artikel auf A4-Papier
- In geschweiften Klammern stehen *obligatorische Argumente*.
Hier: die Art des Dokuments (article, report, book, beamer, dinbrief, ...)

Ein kleines L^AT_EX-Dokument: Analyse

```
1 \documentclass[a4paper]{article}
2 \usepackage[ngerman]{babel}
```

- `\usepackage` Lädt Makropakete für besondere Features.
Hier: Sprachunterstützung für Deutsch (neue Rechtschreibung)

Ein kleines LATEX-Dokument: Analyse

```
1 \documentclass[a4paper]{article}
2 \usepackage[ngerman]{babel}
3 \usepackage[latin1]{inputenc}
```

- Noch ein Makropaket: wir wollen Umlaute im Latin-1-Zeichensatz benutzen.

Ein kleines L^AT_EX-Dokument: Analyse

```
1 \documentclass[a4paper]{article}
2 \usepackage[ngerman]{babel}
3 \usepackage[latin1]{inputenc}
4 \begin{document}
10 \end{document}
```

- `\begin{document}` und `\end{document}` markieren Anfang und Endes des eigentlichen Textes.
- begin/end-Blöcke heißen *Environments* oder Umgebungen.

Ein kleines L^AT_EX-Dokument: Analyse

```
1 \documentclass[a4paper]{article}
2 \usepackage[ngerman]{babel}
3 \usepackage[latin1]{inputenc}
4 \begin{document}
5 \section{Erster Abschnitt}
10 \end{document}
```

- `\section{ }` markiert den Anfang eines Absatzes
- Das Argument gibt den Titel des Absatzes an
- Sections sind das höchste Strukturierungselement eines Artikels

Ein kleines L^AT_EX-Dokument: Analyse

```
1 \documentclass[a4paper]{article}
2 \usepackage[ngerman]{babel}
3 \usepackage[latin1]{inputenc}
4 \begin{document}
5 \section{Erster Abschnitt}
6 Dies ist ein Textabsatz
7 und das hier gehört auch dazu.
8
9 Das ist noch einer.
10 \end{document}
```

- Absätze bestehen aus beliebigen zusammenhängenden Textabschnitten und werden i. d. R. im Blocksatz gesetzt.
- Zeilenvorschübe und Extra-Leerzeichen werden ignoriert; eine Leerzeile beginnt einen neuen Absatz.

Sonderzeichen (1)

Die meisten Zeichen werden in T_EX so gesetzt, wie sie im Quelltext stehen. Wie aus dem Beispiel z. T. schon zu schließen, haben einige aber eine Sonderbedeutung:

- \ Der Backslash leitet ein T_EX-Kommando oder Makro ein. Vor einem Zeichen mit Sonderbedeutung stehend bewirkt er i. d. R., dass dieses Zeichen normal gesetzt wird.
- { } Die geschweiften Klammern gruppieren ihren Inhalt, um ihn z. B. als Makro-Argument zu verwenden.
- % Das Prozentzeichen leitet einen Kommentar ein. Alles auf der selben Zeile dahinter stehende wird ignoriert.

Backslash, Klammern und Prozent

Hervorgehoben und normal

```
\emph{Hervorgehoben}_und_normal_%_Makro_fuer_"emphasis"
```

Sonderzeichen (2)

- § Das Dollarzeichen begrenzt Ausdrücke, die im Mathemodus gesetzt werden sollen.
- ^_ Diese Zeichen bewirken im Mathemodus das Hoch- oder Tiefstellen ihrer Argumente.

Der mathematische Modus

$$c_1 = \sqrt{a_1 + b_1^{10}}$$

```
$c_1 = \sqrt{a_1 + b_1^{10}}$
```

Sonderzeichen (2)

- § Das Dollarzeichen begrenzt Ausdrücke, die im Mathemodus gesetzt werden sollen.
- ^_ Diese Zeichen bewirken im Mathemodus das Hoch- oder Tiefstellen ihrer Argumente.

Der mathematische Modus

$$c_1 = \sqrt{a_1 + b_1^{10}}$$

```
$c_1 = \sqrt{a_1 + b_1^{10}}$
```

- ~ Die Tilde fügt ein Leerzeichen ein, an dem kein Zeilenumbruch erfolgen darf.
- & Im Tabellensatz trennt dieses Zeichen Spalten voneinander.
- # Fügt Argumente in Makrodefinitionen ein.

Strukturierung von L^AT_EX-Dokumenten

Die folgenden Elemente erzeugen Überschriften unterschiedlicher Ordnung und lassen sich automatisch in Inhaltsverzeichnisse aufnehmen:

`\part` Teil eines Buches (nur in Klasse „book“)

`\chapter` Kapitel (nur in „book“ und „report“)

`\section`

`\subsection`

`\subsubsection` Abschnitte (alle Klassen)

`\paragraph` Absatz

Strukturierung von L^AT_EX-Dokumenten

Die folgenden Elemente erzeugen Überschriften unterschiedlicher Ordnung und lassen sich automatisch in Inhaltsverzeichnisse aufnehmen:

`\part` Teil eines Buches (nur in Klasse „book“)

`\chapter` Kapitel (nur in „book“ und „report“)

`\section`

`\subsection`

`\subsubsection` Abschnitte (alle Klassen)

`\paragraph` Absatz

- Die automatische Nummerierung von Abschnitten lässt sich abschalten, in dem der Abschnittsname mit nachgestelltem Stern geschrieben wird, z. B. `\section*{Einleitung}`
- Ein Inhaltsverzeichnis wird mit dem Befehl `\tableofcontents` eingefügt.

Schrifttypen

Als *logische Auszeichnung* kennt L^AT_EX nur das bereits vorgestellte `\emph{}`. Darüber hinaus sind aber verschiedene *physische Auszeichnungen* verfügbar:

`\textrm{...}` Roman-Schriftfamilie

`\textsf{...}` Sans-Serif-Familie

`\texttt{...}` Schreibmaschinenschrift (*typewriter*)

`\textup{...}` Aufrecht (*upright*)

`\textit{...}` *Kursiv (italics)*

`\textsl{...}` *Geneigt (slanted)*

`\textsc{...}` KAPITÄLCHEN (*small caps*)

`\textmd{...}` Normal breit (*medium*)

`\textbf{...}` **Fett (boldface)**

Schriftgrößen

Im Unterschied zu den Schrifttyp-Kommandos, die ihr *Argument* in einem bestimmten Typ setzen, gibt es die Schriftgrößen-Makros nur als „Umschaltkommandos“, die in Gruppierungs-Klammern benutzt werden müssen²⁷

<code>{\tiny ...}</code>	<code>{\large ...}</code>
<code>{\scriptsize ...}</code>	<code>{\Large ...}</code>
<code>{\footnotesize ...}</code>	<code>{\LARGE ...}</code>
<code>{\small ...}</code>	<code>{\huge ...}</code>
<code>{\normalsize ...}</code>	<code>{\Huge ...}</code>

²⁷Was wohl hauptsächlich damit zu tun hat, dass sie eigentlich nicht benutzt werden *sollen*.

Anführungszeichen (1)

Für Anführungszeichen gibt es verschiedene nationale Standards, die in L^AT_EX mit dem babel-Paket unterstützt werden.

Deutsch

Das sind die deutschen „Gänsefüßchen“. Man kann sie auch „so“ schreiben, und bei ‚einfachen‘ muss man das auch.

Hier mal in „Huge“

```
Das_sind_die_deutschen_”“Gänsefüßchen””._Man_kann_sie_auch
\glqq_so\grqq{}_schreiben,_und_bei_\glq_einfachen\grq{}
muss_man_das_auch.\\{\bHuge_Hier_mal_in_”“Huge””}
```

Vorsicht: Ein Leerzeichen nach einem Makroaufruf wird verschluckt! Wenn nach den Anführungszeichen eins stehen soll, muss deswegen mit „{}“ ein leeres Argument eingefügt werden.

Anführungszeichen (2)

Englisch

English does not have “lower” and ‘upper’ quotation marks. Opening and closing marks are still “different” though.

English does not have “lower ” and ‘upper’ quotation marks. Opening and closing marks are still `\Huge` “different ” though.

Französisch

On utilise les guillemets ouvrants (`<<`) suivis d’une espace, et les guillemets fermants (`>>`) précédés d’une espace: `<< guillemets >>`

On utilise les guillemets ouvrants (`\flqq {}`) suivis d’une espace, et les guillemets fermants (`\frqq {}`) précédés d’une espace:
`\flqq {} guillemets \frqq {}`

Umbrüche

Neben der Leerzeile zum beginnen eines neuen Absatzes kann auch ein einfacher Zeilen- oder ein Seitenumbruch eingefügt werden.

Verschiedene Umbrüche

Dies ist ein Textabsatz, der sich zu reinen Demonstrationszwecken etwas hinzieht, obwohl er keinen wichtigen Text enthält.

Er enthält aber immerhin einen Zeilenumbruch (da geht es ohne Einzug weiter!)

Der nächste Absatz beendet dann auch die Seite.

Dies ist ein Textabsatz, der sich zu reinen Demonstrationszwecken etwas hinzieht, obwohl er keinen wichtigen Text enthält. \\

Er enthält aber immerhin einen Zeilenumbruch (da geht es ohne Einzug weiter!)

Der nächste Absatz beendet dann auch die Seite. **\pagebreak**

Listen

- Wie in HTML gibt es in L^AT_EX drei Arten von vordefinierten Listen
 - ① Unnummerierte Listen („bullet-point lists“)
 - ② Nummerierte Listen
 - ③ Definitionslisten
- Alle sind nach dem gleichen Schema mit einer Listenumgebung und dem Kommando `\item` vor jedem Listenelement aufgebaut.
- Das `\item`-Kommando nimmt ein optionales Argument, das den zu verwendenden Listenelement bzw. bei Definitionen den zu definierenden Begriff angibt.
- Die allgemeinere `list`-Umgebung erlaubt eine freiere Definition des Listenelements, benötigt dafür aber mehr Parameter.

Unnummerierte Listen

Itemize

- Das eine

→ Das andere mit einem anderen Listenpunkt und etwas mehr Text, so dass hier umbrochen wird.

- Und ein drittes

```
\begin{itemize}
\item Das eine
\item[$\rightarrow$] Das andere mit einem anderen
Listenpunkt und etwas mehr Text, so dass hier umbrochen wird.
\item Und ein drittes
\end{itemize}
```

Nummerierte Listen

Enumerate

1. Das eine

→ Das andere mit einem anderen Listenpunkt, der zur Folge hat, dass hier nicht weitergezählt wird!

2. Und ein dritter Punkt, der dementsprechend Nr. 2 heißt.

```
\begin{enumerate}
\item Das eine
\item[$\rightarrow$] Das andere mit einem anderen
Listepunkt, der zur Folge hat, dass hier nicht weitergezählt wird!
\item Und ein dritter Punkt, der dementsprechend Nr. 2 heißt.
\end{enumerate}
```

Definitionslisten

Description

itemize Mit dieser Umgebung werden unnummerierte Listen (AKA *bullet-point lists*) gesetzt.

enumerate Diese nummeriert automatisch.

description Umgebung für eine Definitionsliste wie diese.

```
\begin{description}
```

```
\item[itemize] _Mit_ dieser _Umgebung_ werden _unnummerierte_ Listen  
(AKA_ \emph{bullet-point_lists}) _gesetzt_.
```

```
\item[enumerate] _Diese_ nummeriert _automatisch_.
```

```
\item[description] _Umgebung_ für _eine_ Definitionsliste _wie_ diese .
```

```
\end{description}
```

Allgemeine Listen

Das erste obligatorische Argument gibt den zu verwendenden Listenpunkt an, das zweite kann beliebige Änderungen an den Parametern der Listenumgebung²⁸ vornehmen.

List

+ Ein Punkt

+ Und noch einer

```
\begin{list}{\textbf{+}}{\setlength{\labelsep}{1mm}\setlength{\itemsep}{7mm}}\item_Ein_Punkt\item_Und_noch_einer\end{list}
```

²⁸Es gibt deren ein knappes Dutzend, für eine vollständige Liste siehe z. B. Kopka Bd. 1.

Tabulatoren

- Zum setzen von Tabulatoren dient die `tabbing`-Umgebung.
- Innerhalb von `tabbing` kann mit `\=` eine Tabulatorposition gesetzt werden; `\>` rückt zur nächsten Position vor.
- Um die Tabulatoren zu setzen, kann eine „Schablonenzeile“ mit den jeweils längsten Einträgen jeder Spalte gesetzt und mit `\kill` wieder entfernt werden.
- Das Zeilenumbruchzeichen `\\` beendet wie üblich eine Zeile.

Tabulatoren

Spalte1	Spalte2	Spalte3	Spalte4
foo	bar	Blahfasel	baz

```
\begin{tabbing}
Spalte1_\=Spalte2_\=Blahfasel_\=Spalte4\kill
Spalte1\>Spalte2\>Spalte3\>Spalte4\\
foo\>bar\>Blahfasel\>baz
\end{tabbing}
```

Tabellen (1)

- Die `tabular`-Umgebung setzt tabellarische Daten
- Leicht zu verwechseln: `table` umschließt im Text „gleitende“ Tabellen, die ihrerseits i. d. R. `tabular`-Objekte enthalten!

Tabellen (1)

- Die `tabular`-Umgebung setzt tabellarische Daten
- Leicht zu verwechseln: `table` umschließt im Text „gleitende“ Tabellen, die ihrerseits i. d. R. `tabular`-Objekte enthalten!
- `tabular` verlangt eine Spaltendefinition als Argument, die für jede Spalte angibt, wie sie gesetzt werden soll:
 - | Linksbündig
 - r* Rechtsbündig
 - c* Zentriert
 - p{width}* Im Absatzmodus, d. h. mit automatischem Umbruch auf Breite *width*, Text am oberen Zellenrand.
 - | Vertikale Trennlinie zwischen Spalten

Tabellen (2)

- Die Inhalte der einzelnen Spalten werden mit Kaufmannsund (&) voneinander getrennt; das Zeilenumbruchzeichen `\\` beendet eine Tabellenzeile.
- Horizontale Linien werden mit `\hline` nach `\\` eingefügt.

Tabellen (2)

- Die Inhalte der einzelnen Spalten werden mit Kaufmannsund (&) voneinander getrennt; das Zeilenumbruchzeichen `\\` beendet eine Tabellenzeile.
- Horizontale Linien werden mit `\hline` nach `\\` eingefügt.
- Das Paket `array` erlaubt außerdem einige zusätzliche Deklarationen in der Spaltendefinitin:
 - `b{width}` Wie „p“, aber am unteren Zellenrand ausgerichtet.
 - `m{width}` Wie „p“, aber in der Zelle vertikal zentriert.
 - `>{decl.}` Fügt *decl.* vor der folgenden Spaltendefinition ein.
 - `<{decl.}` Fügt *decl.* nach der vorangehenden Spaltendefinition ein.

Tabellen (3)

Einfache Tabelle

Kurs	SWS	Dozent
PS Maschinelle Sprachverarbeitung	2	Besim Kabashi
Übung Grundlagen der CL I	zwei	Jörg Kapfer
Werkzeuge und Arbeitstechniken	auch zwei	Matthias Bethke

```

\begin{tabular}{|r|cl|}
\hline
Kurs_&_SWS_&_Dozent_\\
\hline
\hline
PS_Maschinelle_Sprachverarbeitung_&_2_&_Besim_Kabashi\\
Übung_Grundlagen_der_CL_I_&_zwei_&_Jörg_Kapfer\\
Werkzeuge_und_Arbeitstechniken_&_auch_zwei_&_Matthias_Bethke\\
\hline
\end{tabular}

```

Tabellen (4)

Tabelle mit array-Features

Kurs	SWS	Dozent
PS Maschinelle Sprachverarbeitung	$\text{int}(1.415^2)$	Besim Kabashi
Übung Grundlagen der CL I	$\sqrt{\frac{12}{3}}$	Jörg Kapfer
Werkzeuge und Arbeitstechniken der Computerlinguistik	<i>auch zwei</i>	Matthias Bethke

```

\begin{tabular}{m{.4\textwidth}}|>{\$}c<{\$}r}
Kurs_&_SSWS$&_Dozent_\\
\hline
PS_Maschinelle_Sprachverarbeitung&_int(1.415^2)&_Besim_Kabashi\\
Übung_Grundlagen_der_CL_I_&_sqrt{\frac{12}{3}}&_Jörg_Kapfer\\
Werkzeuge_und_Arbeitstechniken
der_Computerlinguistik_&_auch\_zwei_&_Matthias_Bethke\\
\end{tabular}

```

Dokumententitel

Titel, Autor und Datum

T_EXniken und so

Matthias Bethke

1. Februar 2007

```
\author{Matthias_Bethke}
\title{\TeX{}niken_und_so}
\date{\today}
\maketitle
```

- `\date` ist hier redundant, kann aber für ein leeres Datumsfeld (`\date{}`) oder ein festes oder nicht standardgemäßes Datum (z. B. `\date{Weihnachten_2006}`) benutzt werden.
- Für freier layoutete Titelseiten gibt es die Umgebung `titlepage`.

Eigene Makros (1)

- Wozu überhaupt?
 - Allgemein: Erweitern des Sprachumfangs
 - Logische Stile definieren
 - Häufig gebrauchte Konstruktionen abkürzen
 - Eingebaute Definitionen ändern

Eigene Makros (1)

- Wozu überhaupt?

- Allgemein: Erweitern des Sprachumfangs
- Logische Stile definieren
- Häufig gebrauchte Konstruktionen abkürzen
- Eingebaute Definitionen ändern

- Der Befehl

`\newcommand{name}[narg][default]{definition}`

definiert ein neues Makro.

- `\renewcommand{name}[narg][default]{definition}`

ändert ein bereits definiertes Makro, bzw. überschreibt es mit der neuen Definition.

- Neue Umgebungen erzeugt man mit

`\newenvironment{name}[narg][default]{start-def}{end-def}`.

Eigene Makros (1)

- Wozu überhaupt?
 - Allgemein: Erweitern des Sprachumfangs
 - Logische Stile definieren
 - Häufig gebrauchte Konstruktionen abkürzen
 - Eingebaute Definitionen ändern
- Der Befehl
`\newcommand{name}[narg][default]{definition}`
definiert ein neues Makro.
- `\renewcommand{name}[narg][default]{definition}`
ändert ein bereits definiertes Makro, bzw. überschreibt es mit der neuen Definition.
- Neue Umgebungen erzeugt man mit
`\newenvironment{name}[narg][default]{start-def}{end-def}`.
- Meist empfehlenswert: die Stern-Form der Kommandos (`\newcommand*` etc.) akzeptiert nur „kurze“ Argumente und kann deshalb oft genauere Fehlermeldungen liefern.

Eigene Makros (2)

Logische Stile

„~msbethke“ ist mein Home-Verzeichnis

Ein regulärer Ausdruck: `[[:lower:]].*?\b`

```
\newcommand{\bsl}{\ensuremath{\backslash}}
```

```
\newcommand{\til}{\ensuremath{\sim}}
```

```
\newcommand*{\regex}[1]{\texttt{#1}}
```

```
\noindent“\til_msbethke”_ist_mein_Home-Verzeichnis\
```

```
Ein_regulärer_Ausdruck:.\regex{[[:lower:]].*?\bsl_b}
```

Eigene Makros (2)

Logische Stile

„~msbethke“ ist mein Home-Verzeichnis

Ein regulärer Ausdruck: `[[[:lower:]].*?\b`

```
\newcommand{\bsl}{\ensuremath{\backslash}}
```

```
\newcommand{\til}{\ensuremath{\sim}}
```

```
\newcommand*{\regex}[1]{\texttt{#1}}
```

```
\noindent“\til_msbethke”_ist_mein_Home-Verzeichnis\
```

```
Ein_regulärer_Ausdruck:_\regex{[[[:lower:]].*?\_bsl_b}
```

- Nimmt ein Makro Argumente, muss ihre Anzahl deklariert werden.
- Im Ersetzungstext des Makros (*definition*) werden Argumente mit *#n* eingesetzt.

Eigene Makros (4): ändern vorgegebener Makros

Makros ändern

1 Erster Abschnitt

§ 2 Zweiter Abschnitt

```
\section{Erster_Abschnitt}
```

```
\renewcommand{\thesection}{\S{}~\arabic{section}}
```

```
\section{Zweiter_Abschnitt}
```

Eigene Makros (4): ändern vorgegebener Makros

Makros ändern

1 Erster Abschnitt

§ 2 Zweiter Abschnitt

```
\section{Erster_Abschnitt}  
\renewcommand{\thesection}{\S{}}~\arabic{section}  
\section{Zweiter_Abschnitt}
```

- „\thesection“ ist ein vordefiniertes Makro, das von „\section“ aufgerufen wird und den Abschnittszähler „section“ formatiert.
- Bei solchen vorgegebenen Makros sollten natürlich i. d. R. die Argumente mit denen des Originals (→L^AT_EX-Handbuch) übereinstimmen.

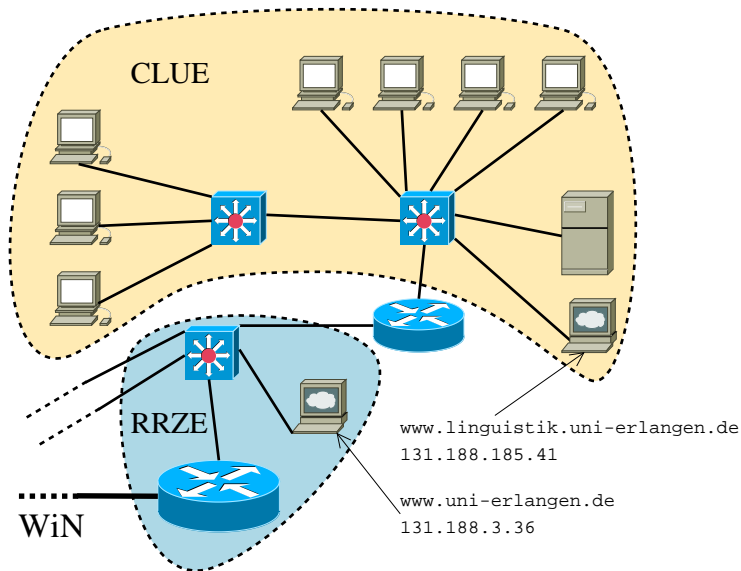
Rechnernetze und TCP/IP

- Das *Netzwerkprotokoll* TCP/IP²⁹ ist die Basis zur Vernetzung aller direkt am Internet beteiligten Rechner.
- Jeder vernetzte Computer hat mit seiner *IP-Adresse* eine weltweit einmalige 32-bit-Nummer.
- Zwecks besserer Lesbarkeit wird diese Nummer üblicherweise in vier Gruppen aufgeteilt und als Dezimalzahlen zwischen 0 und 255 geschrieben, z. B. 131.188.185.41, 128.252.19.2³⁰
- Anhand dieser Adressen werden die in „Pakete“ aufgeteilten Daten im Netz weitergereicht.
- Damit man sich nicht Dutzende von Zahlenfolgen merken muss, assoziiert der *Domain Name Service* symbolische Namen wie `www.linguistik.uni-erlangen.de` mit entsprechenden numerischen Adressen.

²⁹ *Transmission Control Protocol / Internet Protocol*

³⁰ Man trifft derzeit noch selten Adressen nach Version 6 des IP an, die 128 Bit in Hexadezimaldarstellung benutzen, z. B. 1080:0:0:0:8:800:200C:417A

Das CLUE-Netz



Ports und Netzwerkdienste

- Neben der *Adresse* gehört zu jeder Netzwerkverbindung auch ein *Port*, eine Zahl zwischen 0 und 65535, die verschiedene Dienste auf einem Rechner differenziert.
- Einige übliche Portnummern bekannter Dienste
 - 21 File Transfer protocol (`ftp`)
 - 22 Secure Shell (`ssh`)
 - 25 Simple Mail Transfer Protocol (`smtp`)
 - 80 Hypertext Transfer Protocol (`http`)
 - 110 Post Office Protocol V3 (`pop3`)
 - 143 Internet Message Access Protocol (`imap`)
 - 3128 Squid-Webproxies, oft auch 8080
 - 5432 PostgreSQL-Datenbankserver
- Dienste *müssen* nicht die o. a. Ports benutzen, tun dies aber in der Regel.
- Client-Programme benutzen die „übliche“ Portnummer für ihren Dienst, wenn nichts anderes angegeben wird.

Mail

- Normalerweise: menügeführte Mail-Clients (*MUA*, Mail User Agent) wie KMail, Mutt oder Thunderbird.
- Bereits kurz vorgestellt: Shell-Kommando **mail**
- Wichtige Optionen:
 - s <subj> Thema (*Subject*)
 - c <addr>[,addr...] „Durchschlag“ an <addr>
(*Carbon Copy*)
 - b <addr>[,addr...] „Versteckter Durchschlag“ an
<addr> (*Blind Carbon Copy*)
 - a "<text>" Extra-Kopfzeile angeben
 - v Ausführliche Meldungen (*Verbose*)

SSH

- *Secure Shell* ist seit Jahren Quasi-Standard für den sicheren Login auf Systemen im Netz und als Universalwerkzeug für verschlüsselte Verbindungen.
- Früher: *telnet*, übertrug Daten und v. a. Passworte im Klartext → leicht abhörbar.
- Aufruf **ssh** [*Optionen*] [*user@*]*<host>* [*Befehl*]

SSH

- *Secure Shell* ist seit Jahren Quasi-Standard für den sicheren Login auf Systemen im Netz und als Universalwerkzeug für verschlüsselte Verbindungen.
- Früher: *telnet*, übertrug Daten und v. a. Passworte im Klartext → leicht abhörbar.
- Aufruf `ssh [Optionen] [user@]<host> [Befehl]`
- Wichtige Optionen:
 - p Port auf der anderen Maschine
 - X X11-Forwarding, ermöglicht es, grafische Anwendungen auf dem entfernten Rechner laufen zu lassen und lokal zu bedienen.
 - Y Wie -X aber mit weniger Beschränkungen bzgl. Sicherheit. Leider oft benötigt für KDE-/Gnome-Programme.

SSH-Escapes

- Während einer Session werden sämtliche Eingaben 1:1 ans entfernte System übertragen – auch Steuerzeichen wie `^Z`, `^C`, usw.!
- Um an das `ssh`-Programm selbst Kommandos zu geben: *am Zeilenanfang ein Tilde-Zeichen* eingeben (wird nicht angezeigt). Auf dieses Zeichen folgen dann Kommandos wie
 - `^Z` `ssh` stoppen bzw. suspendieren
 - Session beenden (z. B. weil ein Programm hängt und sich nicht beenden lässt)
 - `?` Liste der Tilde-Kommandos ausgeben

PublicKey-Authentisierung

- Alternative zur Passwort-Authentisierung, benutzt eine Schlüsseldatei in `~/.ssh`
- Schlüsseldatei kann zusätzlich mit einer Passphrase geschützt werden.

PublicKey-Authentisierung

- Alternative zur Passwort-Authentisierung, benutzt eine Schlüsseldatei in `~/ssh`
- Schlüsseldatei kann zusätzlich mit einer Passphrase geschützt werden.
- Schlüsseldatei erzeugen: `ssh-keygen -trsa` (Vorgaben akzeptieren, Passphrase kann beim Einsatz nur innerhalb der CLUE leer bleiben)
- Läuft ssh-agent? `ps ax | grep agent`
Falls nicht: `eval $(ssh-agent)`
- `ssh-add` sollte melden „Identity added: ...“
- Anschließend: `ssh-copy-id Rechnername`
(verlangt noch einmal das Login-Passwort)
- Ergebnis: man kann sich auf allen CLUE-Rechnern ohne Passwort einloggen.

scp

- *scp* ist ein Werkzeug aus der *ssh*-Suite, das Dateien sicher übers Netz kopiert.
- Benutzung: weitgehend wie *cp*, aber mit der Möglichkeit, Rechnernamen mit anzugeben:
`scp [[user@]host1:]file1 ... [[user@]host2:]file2`
- **Vorsicht:** Port-Angabe mit `-PPort` statt `-pPort` wie bei *ssh*!

scp

- *scp* ist ein Werkzeug aus der *ssh*-Suite, das Dateien sicher übers Netz kopiert.
- Benutzung: weitgehend wie *cp*, aber mit der Möglichkeit, Rechnernamen mit anzugeben:
`scp [[user@]host1:]file1 ... [[user@]host2:]file2`
- **Vorsicht:** Port-Angabe mit `-PPort` statt `-pPort` wie bei *ssh*!

Aus /tmp auf clue14 nach ~/Files

```
msbethke@clue45 ~ $ scp clue14:/tmp/myfile ~/Files
myfile                100%  399      0.4KB/s   00:00
```

Anderer Account auf fremdem Rechner

```
msbethke@clue45 ~ $ scp mb@drgonzo.dyndns.org:myfile .
```

FTP (1)

- Das *File Transfer Protocol* dient, wie der Name sagt, zum Dateitransfer zwischen Rechnern.
- Heutzutage oft durch *scp* ersetzt, aber v. a. für anonymen Transfer großer Dateien immer noch sehr beliebt.
- Per Webbrowser: `ftp://...-URLs`
- Kommandozeilenclients: *ftp*, *ncftp*
- Aufruf:
`ftp [[user@]host [port]]`
`ncftp [-u user] [-P port] [-p pass] [host|URL]`
- Wichtigste Kommandos: **dir**, **cd**, **get**, **put**, **help**,

FTP (2)

FTP-Session

```
msbethke@clue14:~> ncftp ftp.uni-erlangen.de
NcFTP 3.1.9 (Mar 24, 2005) by Mike Gleason
Connecting to 131.188.3.71...
----- Welcome to Pure-FTPd [TLS] -----
You are user number 598 of 800 allowed.
Local time is now 13:36. Server port: 21.
Only anonymous FTP is allowed here
[...]
ncftp / > dir
drwxrwxr-x          Jan 23 09:56   debian
drwxr-xr-x          Jan 23 04:38   debian-amd64
drwxr-xr-x          Jan 23 03:52   debian-cd
-rw-r--r--          537   Feb  3  2005  README.MOUNTS
ncftp / >
```