Einführung in das Textsatzsystem

LaTeX

03 – Formatieren und Definieren

Teaching the Lion

8. November 2013

Inhalt

TeXworks
Nützliche Feinheiten

- Einstellung der möglichen Tools:
  `Edit` ↦ `Preferences` ↦ `Typesetting`

TeXworks
Nützliche Feinheiten

- Einstellung der möglichen Tools:
  `Edit` ↦ `Preferences` ↦ `Typesetting`

- Autovervollständigung: Viele Befehle sind bekannt und können vervollständigt werden:
  `\doc <tab>` ⇒ `\documentclass{}`

`bit <tab>` ⇒ `\begin{itemize} \end{itemize}`

- bei Mehrdeutigkeit: mehrmals `<tab>` drücken
- Einstellbar über Datei `tw-latex.txt`, meist im Nutzerordner unter `./TeXworks/completion/tw-latex.txt`

Formatierung: Makrotypographie

- „globale“ Typographie:
- Satzspiegel
- Gestaltung von Kopf- und Fußzeilen
- Wahl der Schriften
- Formatierung von Abständen u. ä.
- Inhaltsverzeichnis, Indizes, Verzeichnisse …

Der Satzspiegel

Bezeichnet alles, was auf der Seite zum Bedrucken genutzt wird:

- Verhältnis von Rändern zu Textbreiten
- ein- oder zweiseitiger Satz
- ein- oder mehrspaltiger Satz
- Kopf- und Fußzeilen
- Satzspiegel abhängig von Schriftgröße und Laufweite, für ein harmonisches Bild

Der Satzspiegel

Bezeichnet alles, was auf der Seite zum Bedrucken genutzt wird:

- Verhältnis von Rändern zu Textbreiten
- ein- oder zweiseitiger Satz
- ein- oder mehrspaltiger Satz
- Kopf- und Fußzeilen
- Satzspiegel abhängig von Schriftgröße und Laufweite, für ein harmonisches Bild

Leider ist der Satzspiegel meist (vom Herausgeber, Professor, …) vorgegeben, oft mit typographisch fragwürdigen Einstellungen – zu kleine oder unpassende Ränder, unharmonische Seitenaufteilung etc.

Satzspiegel mit KOMA

Als Alternative zu den Standardklassen bietet das KOMA-Skript (Hauptautor Markus Kohm):

- optimale Satzspiegelkonstruktion
- mit dem speziellen Paket `typearea`
- automatische Berechnung (Paket laden, fertig)
- anpassbar für Spezialfälle (z. B. mittelalterliche Satzspiegelkonstruktion)

Satzspiegel mit KOMA

Als Alternative zu den Standardklassen bietet das KOMA-Skript (Hauptautor Markus Kohm):

- optimale Satzspiegelkonstruktion
- mit dem speziellen Paket `typearea`
- automatische Berechnung (Paket laden, fertig)
- anpassbar für Spezialfälle (z. B. mittelalterliche Satzspiegelkonstruktion)

Falls das alles nicht gefällt: freie Anpassung aller Dimensionen mittels `geometry` (unabhängig von KOMA)

Das Paket geometry

- erlaubt manuelle Einstellung des Satzspiegels
- alle Parameter werden mittels Paketoption angegeben:
  `\usepackage[top=2cm,bottom=5cm]{geometry}`
- *oder* mittels `\geometry{top=2cm,bottom=5cm}`
- mittels `\newgeometry` kann der Satzspiegel geändert werden
- `\restoregeometry` stellt vorherige Einstellungen wieder her
- `\savegeometry{}` und `\loadgeometry{}` erlauben Speichern und Laden von Einstellungen

Das Paket geometry

Mögliche Optionen (kleine Auswahl):

```
paperheight, paperwidth
left, right, inner, outer, hmargin
top, bottom, vmargin
bindingoffset, textwidth, textheight
twoside, twocolumn, columnsep
marginparsep, footnotesep, headsep, footsep, nofoot, nohead
hoffset, voffset, offset
```

⁝⟹ siehe `texdoc geometry` für ausführliche Beschreibung und Dokumentation

Kopf- und Fußzeilen

- Enthalten wichtige Informationen über das Dokument:
- Lebende Kolumnentitel (Kapitel-/ Abschnittsüberschriftend)
- Seitenzahlen
- evtl. kleine Verzierungen, Schnörkel, …

Kopf- und Fußzeilen

- Enthalten wichtige Informationen über das Dokument:
- Lebende Kolumnentitel (Kapitel-/ Abschnittsüberschriftend)
- Seitenzahlen
- evtl. kleine Verzierungen, Schnörkel, …

- Verschiedene Pakete erlauben Anpassungen
- Auswahl des Seitenstils mittels
  `\pagestyle{Seitenstil}` (für alle Seiten) oder
  `\thispagestyle{Seitenstil}` (nur diese Seite)
- Voreinstellungen im LaTeX-Kernel:
  `empty`, `plain`, `headings`

Seitenstil mit KOMA: scrpage2

```
\usepackage[automark]{scrpage2}
\pagestyle{scrheadings} %% entweder scrheadings oder
scrplain
\pagestyle{scrplain} %%    muss aktiviert werden
\lehead[scrplain-links-gerade]{scrheadings-links-gerade}
\cohead[scrplain-zentriert-ungerade]{scrheadings-z-u}
\rehead[]{}
\cefoot[]{}
\ohead[]{\pagemark}
\ihead{\headmark}
\cfoot{}
```

$\Longrightarrow$ siehe `texdoc scrguide`

Seitenstil anpassen mit `fancyhdr`

Laden des Pakets und Einstellungen im Dokumentenkopf:

```
\usepackage{fancyhdr}
\pagestyle{fancy} %% ohne das funktioniert nichts
```

Einseitiger Satz:

```
\lhead{}    \lfoot{}
\chead{}    \cfoot{}
\rhead{}    \rfoot{}
```

Seitenstil anpassen mit `fancyhdr`

Laden des Pakets und Einstellungen im Dokumentenkopf:

```
\usepackage{fancyhdr}
\pagestyle{fancy} %% ohne das funktioniert nichts
```

Einseitiger Satz:

```
\lhead{}    \lfoot{}
\chead{}    \cfoot{}
\rhead{}    \rfoot{}
```

Zweiseitiger Satz (Odd, Even)

```
\fancyhead[LO]{}
\fancyhead[RO,LE]{}
\fancyhead[CE]{}
\fancyfoot[LO]{}
\fancyfoot[RO,LE]{}
\fancyfoot[CE]{}
```

Schriftgrößen

Dokumentoption

Fast alle Klassen bieten Optionen für die Standardschriftgröße:
`\documentclass[10pt]{scrartcl}`

abgeleitete Größen

Davon abgeleitet sind alle anderen Größen (`\tiny`, `\small`, ...), speziell `\footnotesize` und `\scriptsize`.
$\Longrightarrow$ vordefinierte Schriftgrößen erlauben konsistentes Layout.

Schriftgrößen

Dokumentoption

Fast alle Klassen bieten Optionen für die Standardschriftgröße:
`\documentclass[10pt]{scrartcl}`

abgeleitete Größen

Davon abgeleitet sind alle anderen Größen (`\tiny`, `\small`, ...), speziell `\footnotesize` und `\scriptsize`.
⟹ vordefinierte Schriftgrößen erlauben konsistentes Layout.

Wer wirklich *genau weiß*, was er tut:

```
\fontsize{Größe}{Durchschuss}\selectfont
\fontsize{10}{12}\selectfont
```

Typographisch korrekte Strichlängen

Leerzeichen und Striche

Je nach Zweck, Position und Bedeutung werden verschiedene horizontale Striche und Leerräume verwendet.

Typographisch korrekte Strichlängen

Leerzeichen und Striche

Je nach Zweck, Position und Bedeutung werden verschiedene horizontale Striche und Leerräume verwendet.

- Grammatik und Lesbarkeit bestimmen die nötige Strichlänge
- Zu unterscheiden:
- Viertelgeviertstrich oder Bindestrich: `-`
- Halbgeviertstrich oder Gedankenstrich: `--` –
- Geviertstrich oder englischer Gedankenstrich: `---` —
- Minuszeichen: `$-$` −

- Im Vergleich:

`-` `--` `---` `$-` `+$` (Eingabe, falls keine Unicodeeingabe)

- – — −+ (Ausgabe)

Typographisch korrekte Strichlängen

Verwendung

Vorder- und Rückseite …
Vorderseite – oder auch Rückseite – …
frontmatter—or backmatter— …
Längen/Zeitangaben: Düsseldorf – Koblenz; 18–20 Uhr
Unterschiedliche Empfehlungen für das Setzen von Spatia (kleine Leerzeichen, siehe unten)

Leerräume

- normales Leerzeichen: ␣, erzwungenes Leerzeichen: `\␣`, nichttrennbares Leerzeichen: `~`
- schmales Leerzeichen (Spatium): `\,`
- negativer Abstand: `\!`
  vergleiche normal, schmal, negativ: z. B., z.\,B., z\!B.
- großer Abstand (Geviert) `\emspace`
- beliebiger Platz: `\hspace(*){2em}`
- Leerräume im Mathemodus:
  (werden normal automatisch korrekt gesetzt)
  `$a b = a\, b = a \! b$`
  $ab = a\,b = a\!b$

Leerräume

- normales Leerzeichen: ␣, erzwungenes Leerzeichen: `\␣`, nichttrennbares Leerzeichen: `~`
- schmales Leerzeichen (Spatium): `\,`
- negativer Abstand: `\!`
  vergleiche normal, schmal, negativ: z. B., z.B., zB.
- großer Abstand (Geviert) `\emspace`
- beliebiger Platz: `\hspace(*){2em}`
- Leerräume im Mathemodus:
  (werden normal automatisch korrekt gesetzt)
  `$a b = a\, b = a \! b$`
  $ab = a\,b = a\!b$

- Explizites Ändern des Abstandes (kerning):
  `a\kern-0.1em b` ⟹ ab
  `a\kern-0.5em b` ⟹ b

Auslassungen

Auslassungspunkte sollten gesperrt gesetzt werden:
... (falsch)
… (richtig)
Wenn ganze Satzteile …
ausgelassen werden … setzt man Leerzeichen.
Wenn Wortteile ausgelassen werden, kommt kein Leer… davor.

Befehle `\dots` oder `\ldots` sorgen für richtige Abstände.
Paket `ellipsis` korrigiert den Abstand vor und nach Ellipsen.
Achtung, fontspec: bei Verwendung von fontspec *muss* das ellipsis-Paket (*nach* fontspec) geladen werden, da sonst falsche Abstände resultieren.

Auslassungen

Auslassungspunkte sollten gesperrt gesetzt werden:
... (falsch)
… (richtig)
Wenn ganze Satzteile …
ausgelassen werden … setzt man Leerzeichen.
Wenn Wortteile ausgelassen werden, kommt kein Leer… davor.

Befehle `\dots` oder `\ldots` sorgen für richtige Abstände.
Paket `ellipsis` korrigiert den Abstand vor und nach Ellipsen.
Achtung, fontspec: bei Verwendung von fontspec *muss* das ellipsis-Paket (*nach* fontspec) geladen werden, da sonst falsche Abstände resultieren.

Pro-Tip

Ausnutzen aktiver Zeichen (nur mit Vorsicht!)

```
\catcode`…=13
```

Probleme mit `ellipsis`

`\usepackage{ellipsis}` führt zu inkonsistenten Abständen nach `\dots`:

```
\LaTeX ist \dots manchmal kompliziert \dots
```

LaTeXist …manchmal kompliziert …

Probleme mit `ellipsis`

`\usepackage{ellipsis}` führt zu inkonsistenten Abständen nach `\dots`:

```
\LaTeX ist \dots manchmal kompliziert \dots
```

LaTeXist …manchmal kompliziert …

`\usepackage[xspace]{ellipsis}` korrigiert dies:

```
\LaTeX\ ist \dots manchmal kompliziert \dots
```

LaTeX ist … manchmal kompliziert …

Welche Befehle möglich?

- Woher weiß man, welche Befehle `ellipsis` oder `geometry` bieten?
- Welche Einstellungen haben die KOMA-Klassen?
- Wie bekommt man allgemein Informationen über Pakete / Klassen?

Dokumentationen mittels texdoc

- `texdoc` durchsucht die LaTeX-Ordner
- liefert direkt die Dokumentation des gesuchten Paketes:
- `texdoc amsmath` öffnet `amsmath.pdf`
- `texdoc -l amsmath` listet alle Ergebnisse auf
- `texdoc -s amsmath` liefert Ergebnisse aus erweiterter Suche
- `texdoc --help` bietet weitere Informationen

$\Rightarrow$ Vorteile der Kommandozeile

`texdoctk` bietet graphische Oberfläche mit Auswahlmenüs – gut zum „Blättern“, wenn man nicht weiß, was man sucht.

Makros

- Abkürzungen, die das Leben erleichtern
- LaTeX definiert große Zahl von Makros vor
- Klassen und Pakete erweitern die Möglichkeiten immens
- für den Eigenbedarf kann der Nutzer selbst neue Makros definieren
- Makros nehmen Argumente an, die mandatorisch oder optional sein können.
- Argumente sind einzelne Zeichen (genauer, Token) oder Gruppen `{abc}` in geschweiften Klammern.
- Optionale Argumente meist in eckigen Klammern, aber Paketautoren sind kreativ: runde, spitze Klammern u. ä. kommt vor.

Neue Befehle mit xparse

- LaTeX3 bietet durch das xparse-Paket hervorragende Syntax für neue Befehle
- für *Dokumentenebene* geeignet (also Paketautoren und Dokumentautoren)
- `\NewDocumentCommand\cmd{args}{def}`
- Makro ohne Argumente:
  `\NewDocumentCommand\cmd{}{Code}` setzt den angegbenen Code.

Neue Befehle mit xparse

- LaTeX3 bietet durch das xparse-Paket hervorragende Syntax für neue Befehle
- für *Dokumentenebene* geeignet (also Paketautoren und Dokumentautoren)
- `\NewDocumentCommand\cmd{args}{def}`
- Makro ohne Argumente:
  `\NewDocumentCommand\cmd{}{Code}` setzt den angegbenen Code.

- ein Argument:
  `\NewDocumentCommand\cmd{m}{Text: #1}`

- `m` für `mandatory` – muss vorhanden sein.
- Argument kann mittels `#1` verwendet werden.

Argumenttypen in `xparse` (Auswahl)

m	mandatory – muss gegeben werden, in geschweiften Klammern
o	optional – kann vorhanden sein oder nicht, in eckigen Klammern
O{default}	dito, mit Voreinstellung
d<>	delimiter – ebenfalls optional, mit den angegebenen Begrenzungen `\cmd<a>`
D(){default}	dito, mit Voreinstellung, und Begrenzenungen `()`
g/G	Spezialfall für `d/D{}`
t{X}	testet, ob das Token `X` an der angegeben Stelle steht. (boolean)
s	Sternversion, Spezialfall von `t{*}`
u{X}	liest ein Argument bis (until) dem angegebenen Token ein, mandatorisch
l	liest bis zur ersten öffnenden `{`, mandatorisch

Neue Befehle „aus dem Antiquariat“

- TeX-Basisbefehl zum definieren neuer Makros: `\def`
- LaTeX definiert flexiblere Definitionen mit speziellen Absicherungen
- LaTeX 2_ε: `\newcommand`, `\renewcommand` und `\declarecommand`

```
\newcommand[args]\name{definition}
\newcommand[opt][2]\mycommand{#1, #2}
\mycommand{ab}{cd}
```
$\longrightarrow$ ab, cd

Leerzeichen in TeX

Achtung:

TeX „frisst“ gerne Leerzeichen – vor allem nach Befehlen:

`\wasser ist nass` ⇒ H_2Oist nass.

Leerzeichen in TeX

Achtung:

TeX „frisst“ gerne Leerzeichen – vor allem nach Befehlen:

`\wasser ist nass` ⟹ H_2Oist nass.

- TeX liest Befehle vom `\` bis zum ersten nicht-Buchstaben (Zahl, Klammer, Leerzeichen, Punkt, …)
  `\LaTeX ist manchmal umständlich`

LaTeXist manchmal umständlich

Leerzeichen in TeX

Achtung:

TeX „frisst“ gerne Leerzeichen – vor allem nach Befehlen:

`\wasser ist nass` ⟹ H_2Oist nass.

- TeX liest Befehle vom `\` bis zum ersten nicht-Buchstaben (Zahl, Klammer, Leerzeichen, Punkt, …)
  `\LaTeX ist manchmal umständlich`

- Befehle im Text immer mit `\` oder `{}` beenden:

LaTeXist manchmal umständlich

Leerzeichen in TeX

Achtung:

TeX „frisst“ gerne Leerzeichen – vor allem nach Befehlen:

`\wasser ist nass` ⟹ H_2Oist nass.

- TeX liest Befehle vom `\` bis zum ersten nicht-Buchstaben (Zahl, Klammer, Leerzeichen, Punkt, …)
  `\LaTeX ist manchmal umständlich`

- Befehle im Text immer mit `\` oder `{}` beenden:
- `\LaTeX\ ist manchmal umständlich.`

LaTeX ist manchmal umständlich

Leerzeichen in TeX

Achtung:

TeX „frisst“ gerne Leerzeichen – vor allem nach Befehlen:

`\wasser ist nass` ⟹ H_2Oist nass.

- TeX liest Befehle vom `\` bis zum ersten nicht-Buchstaben (Zahl, Klammer, Leerzeichen, Punkt, …)
  `\LaTeX ist manchmal umständlich`

- Befehle im Text immer mit `\` oder `{}` beenden:
- `\LaTeX\ ist manchmal umständlich.`

LaTeX ist manchmal umständlich

Leerzeichen in TeX

Achtung:

TeX „frisst“ gerne Leerzeichen – vor allem nach Befehlen:

`\wasser ist nass` ⇒ H_2Oist nass.

- TeX liest Befehle vom `\` bis zum ersten nicht-Buchstaben (Zahl, Klammer, Leerzeichen, Punkt, …)
  `\LaTeX ist manchmal umständlich`

- Befehle im Text immer mit `\` oder `{}` beenden:
- `\LaTeX\ ist manchmal umständlich.`

LaTeX ist manchmal umständlich

⇒ um konsistent zu werden, ignoriert LaTeX3 auf Paketebene *alle* Leerzeichen!

xspace

- Bei selbstdefinierten Befehlen: `\xspace` aus `xspace`
- `\NewDocumentCommand\wasser{}{HO\xspace}`
- `\xspace` fügt Leerzeichen da ein, wo es sinnvoll ist, lässt es aber weg, wo es nicht hingehört.
  $\Longrightarrow$ ganz gewöhnliches Schreiben ist möglich.

xspace

- Bei selbstdefinierten Befehlen: `\xspace` aus `xspace`
- `\NewDocumentCommand\wasser{}{HO\xspace}`
- `\xspace` fügt Leerzeichen da ein, wo es sinnvoll ist, lässt es aber weg, wo es nicht hingehört.
  $\Longrightarrow$ ganz gewöhnliches Schreiben ist möglich.

$\Longrightarrow$ `\wasser ist nass. \wasser, gefroren, ist nicht nass.`

$\Longrightarrow$ H_2O ist nass. H_2O, gefroren, ist nicht nass.

xspace

- Bei selbstdefinierten Befehlen: `\xspace` aus `xspace`
- `\NewDocumentCommand\wasser{}{HO\xspace}`
- `\xspace` fügt Leerzeichen da ein, wo es sinnvoll ist, lässt es aber weg, wo es nicht hingehört.
  ⟹ ganz gewöhnliches Schreiben ist möglich.

⟹ `\wasser ist nass. \wasser, gefroren, ist nicht nass.`

⟹ H_2O ist nass. H_2O, gefroren, ist nicht nass.

- Für Spezialfälle (besondere Satzzeichen, Sonderzeichen, ?...):
  Anpassung möglich mittels `\xspaceaddexceptions{א}`
  ⟹ `\wasser א` ⟹ H_2O א

Low Level: def

- TeX kennt nur den primitiven Befehl `\def`
- low level ⟹ Vorsicht!
- `xparse` definiert `\NewDocumentCommand` aus guten Gründen! (schützt vor Überschreiben)
- für spezielle Ansprüche kann `\def` aber nützlich sein:

```
\def\dosomething#1#2xyz#3{some code}
```

(mit `xparse` auch möglich, aber längere Definition)