

# (n)vim+L<sup>A</sup>T<sub>E</sub>X



Einrichtung und Anwendung  
Xaver Haider

300.951, KV Wissenschaftliches Schreiben und Layouten anhand von L<sup>A</sup>T<sub>E</sub>X 2

2024-09-18

1. Diese Präsentation ist **kein detailliertes vim-Tutorial**. Sie schafft lediglich einen Einblick auf die Möglichkeiten des Editors und streift dabei grundlegenden Konzepte.
2. Die folgenden Folien fassen wichtigste Punkte zusammen und werden von **Demonstrationen im Editor** begleitet.
3. Das angeführte **Repository** enthält sämtliche Konfigurationsdateien und eine ausführliche Dokumentation in Form einer **Seminararbeit** (Details zu Einrichtung und zusätzliche Informationen):

 xvr0/nvim-latex

[github.com/xvr0/nvim-latex](https://github.com/xvr0/nvim-latex)

# Voraussetzungen

Für die vollständige Einrichtung erforderliche Pakete  
(getestet mit angeführter Versionen):

- Editor: neovim 0.10.1
- Compiler(T<sub>E</sub>X): texlive-\* 2024.2
- Compiler(config): lua 5.4.7
- Terminal: alacritty 0.13.2
- PDF-Viewer: zathura 0.5.8
- Languageserver: texlab 4.3.2 (cargo oder npm)
- Für Plugins: node.js 22.7.0
- Compiler(C): gcc 14.2.1

Stabil unter zahlreichen GNU-Linux Operationssystemen (und Windows).

# Inhalt

- Einführung und Konfiguration
- Nützliche Funktionen
- Anwendungsbeispiele

# Einführung und Konfiguration



# neovim<sup>2</sup>

Nachfolger und Erweiterung (drop-in) von **vim**<sup>1</sup>.

- Tastengesteuert: vi-Keybindings
- Leicht und schnell, dennoch zahlreiche Funktionen *“out of the box”*
- Hoch Konfigurierbar mit *vim-script* oder *lua*
- Starke Unterstützung und zahlreiche Erweiterungen.

<sup>1</sup><https://www.vim.org/>

<sup>2</sup><https://neovim.io/>

## vi-Keybindings (1)

Steile Lernkurve - aber signifikante Steigerung der Arbeitsgeschwindigkeit möglich.

### **Tastengesteuert mit 3 Modi:**

- **INSERT:** Tasten erfüllen ihre Standardfunktion.
- **NORMAL:** Tasten mit Funktionen belegt.
- **VISUAL:** Spezieller Modus zur Auswahl und Bearbeitung von Textstellen.

Wechsel von NORMAL: INSERT: i, VISUAL: v

Zurück zu NORMAL: ESC

*vi-Keybindings* in zahlreichen Umgebungen als Plugin oder Einstellung verfügbar.

## vi-Keybindings (2)

Es gibt zahlreiche **NORMAL**-Tastenfunktionen. Viele werden hier nicht aufgelistet!  
:help für lokale Dokumentation!

- **Navigation** (:help navigation) mit h j k l, gg G: Dateianfang -ende, w W b B: Wortende -anfang, 0, \$: Zeilenanfang -ende, { } ( ): Paragraphen
- **Manipulation** (:help editing): d elete, y ank, p aste  
Wenn groß bis ans Ende der Zeile, doppelt für ganze Zeile (D Y dd yy).  
Speicher in Register: Ohne Angabe Standard sonst z.B.: "ayy -kopiert Zeile in Register a. (Register + für Clipboard reserviert)

Alle Befehle mehrfach ausführbar: 10dd -löscht (und speichert) nächsten 10 Zeilen.  
Nach einfacher Manipulation muss Reichweite gegeben werden: dw löscht bis an Wortende, i für ganzes Wort: diw



## vi-Keybindings (3)

- : allgemein für **Kommandozeile** (:! <shell> - bash-Befehle) (:help cmdline):
  - :w :q :wq -schreiben, schließen oder beides mit ! für force
  - :e <file> öffnet Datei
  - :Ex -öffnet integrierten Explorer
  - :term -öffnet integriertes Terminal
- **Suche**: f<char> Cursor auf Treffer in Zeile  
/<string>ENTER (? -rückwärts) über Datei: navigation mit n, N
- Wechsel in **Einfügemodus** (:help editing)
  - i a -vor oder nach Cursor,
  - c -(change) d und i (Funktion wie d y, z.B.: 10cc ciW)
- :u <C-r> - undo redo

Tipp: Verwendung eines Schummelzettels (vim-cheatsheet)

# Konfiguration

Unterstützt elegante und schnelle Skriptsprache **Lua**<sup>3</sup>.  
Details in der offiziellen Onlinedokumentation *Lua in Nvim*<sup>4</sup>.

- Lädt Wurzel `init.lua` aus `/.config/nvim`
- Struktur über Dateien möglich mit `require("<file>")`
- Einfache Einbindung komplexer Funktionen und Plugins

## Beispiele

```
vim.wo.number= true  
vim.keymap.set("n","<leader>fs","<cmd> TexlabForward) <CR>")
```

<sup>3</sup><https://www.lua.org/>

<sup>4</sup><https://neovim.io/doc/user/lua.htm>

# Plugins

Zahlreiche und umfangreiche Erweiterungen im Rahmen von open-source-Projekten dank starker Nutzergemeinschaft.


- Eingebunden mit `vimscript` oder **Lua**
- Direkt installieren (`git clone`) oder über **Pluginmanager** (z.B: *packer*)

Neovim verfügt über eingebauten Parser und LSP. Alternativen und Erweiterungen bieten zusätzliche Funktionen:


- *tree sitter*: **Parser**-Bibliothek implementiert in C. Updated inkrementell und ist zudem akkurater und schneller als der nvim-Standard.
- *texlab*<sup>5</sup>: **Languageserver** für  $\text{\LaTeX}$ -Dateien (nicht vim spezifisch). lsp-config automatisiert einen Großteil der LSP-Konfiguration.
- *nvim-cmp*: **Completion**-Engine implementiert in Lua. Fertige Bibliotheken oder benutzerspezifische Vorschläge und Snippets können eingebunden werden.

<sup>5</sup><https://github.com/latex-lsp/texlab>

# Github-Repositoryn ausgewählter Erweiterungen

 **wbthomason/packer**

[github.com/wbthomason/packer](https://github.com/wbthomason/packer)

 **nvim-treesitter/nvim-treesitter** [github.com/nvim-treesitter/nvim-treesitter](https://github.com/nvim-treesitter/nvim-treesitter)

 **neovim/nvim-lspconfig**

[github.com/neovim/nvim-lspconfig](https://github.com/neovim/nvim-lspconfig)

 **hrsh7th/nvim-cmp**

[github.com/hrsh7th/nvim-cmp](https://github.com/hrsh7th/nvim-cmp)

 **pwmt/zathura-pdf-poppler**

[github.com/pwmt/zathura-pdf-poppler](https://github.com/pwmt/zathura-pdf-poppler)

# Nützliche Funktionen



Parser und LSP-Support ermöglichen typischen **IDE-Funktionen** wie:

- Syntax-Highlights (Parser)
- Warnungen und Fehlermeldungen (LSP und Compiler (Texlive))
- Code-Refactoring (Parser)
- Snippets und Autocomplete
- PDF-Vor- und Rückwärtssuche (über LSP, Zathura)
- Rechtschreibkorrektur und Thesaurus (Luaconfig)
- Terminal und Explorer (Integriert)

Stärke des Editors in zusätzlichen **vim-spezifischen Operationen**:

- search and replace (:substitute)
- macros
- buffer

# search and replace

Substituiert Muster über ausgewählten Bereich. :help (:substitute)

:<range>s/<string1>/<string2>/<optionen>

## ■ :help range

Standardwert des optionalen Parameters ist die aktuelle Zeile, % wendet den Befehl auf die gesamte Datei an, ., .+<Zeilen> gibt den Bereich von der aktuellen Zeile (.) bis zur Zeile (.+<Zeilen>) an. Die Zeilennummern können auch absolut angegeben werden (<start>,<stop>).

## ■ optionen

g wenn der Befehl auf alle Instanzen anzuwenden ist (Standard: erste der Zeile), i um Groß- und Kleinschreibung zu ignorieren, c um jede Instanz zu prüfen (y or n prompt).

## ■ string

Das zu ersetzende Muster wird zuerst angeführt. Der Syntax umfasst eine Vielzahl an Möglichkeiten komplexere Muster zu erfassen.

- **Minimalsyntax** :s/foo/bar  
Ersetzt die erste Instanz des Strings foo in aktuellen Zeile mit bar .
- **JKU Ganze Wörtern**: :. ,.+5s/\bwort\b/neu/g  
Ersetzt das Wort wort mit neu in der aktuellen und den folgenden 5 Zeilen.  
Partielle Treffer wie wortlaut werden nicht berücksichtigt.
- **Wortgruppen**: :%s/\<(wort1\) and \<(wort2\)\/\2 and \1/g  
Die Klammern fassen die Teilmuster wort1, wort2. Die Muster werden temporär (Variablen: 1 und 2) gespeichert und können im Ersatztext eingefügt werden.
- **Ziffern und chars**: :%s/\Das Muster \



- **SOL und EOL:** `:%s/^wort/neu/g`, `:%s/wort$/neu/g`

^ referenziert den Beginn einer Zeile, \$ das Ende. Das Wort wird in diesem Beispiel nur ersetzt, wenn es sich an der angegebenen Position befindet. Diese Sonderzeichen können auch alleine angegeben werden: `:s/^/%` kommentiert eine Zeile in  $\text{\LaTeX}$ -Dokumenten aus.

- **Wildcard:** `:%s/w.r.t/neu/g`

Der Punkt `.` symbolisiert eine Wildcard. `wort`, `wirt` und `w0rt` werden hier erfasst.

- **Einrückungen:** `:%s/\t/ /g`

`:s/^ \s\+/space`

`\t` ist das Sonderzeichen für Tabulatoren. Leerzeichen und Tabulatoren werden von vim unterschieden, doch unter dem Symbol `\s` zusammengefasst. Das zweite Beispiel zeigt, wie alle Einrückungen zu Beginn einer Zeile ersetzt werden können.

- **Wortwiederholung:** `:%s/\\(\\w\\+\\) \\1/\\1/g`

Hier werden einige bereits beschriebene Konzepte kombiniert. Neu ist die Wildcard für Buchstaben `\\w`. `\\(\\w\\+\\)` erfasst Buchstaben zu einer Gruppe (Wort). `\\1` referenziert diese. Zwei idente aufeinanderfolgende Worte führen zu einem Treffer und werden hier auf ein Wort reduziert.

- **Zeilenumbrüche:** `:%s/\\n\\{2,}\\r/g`

`\\n` ist erwartungsgemäß das Sonderzeichen für newline. `(\\n\\{2,\\})` sucht nach mindestens 2 aufeinanderfolgenden Umbrüchen und ersetzt diese durch einen einzigen `\\r` (carriage return)). Ein weiteres nützliches Werkzeug für die Arbeit an langen Dokumenten.

# macros

Vim bietet die Möglichkeit **Befehlsfolgen** in Registern zu speichern und automatisch auszuführen:

Aufzeichnen: `q<register><Befehlskette>q`

Wiedergabe: `<(integer)>@<register>`

- Keine Einschränkung der Funktionalität
- Beenden mit q  $\Rightarrow$  keine Verschachtelung möglich
- Einfache Handhabung (kein neuer Syntax) und direktes Feedback

### ■ Einrücken oder Kommentieren

`:. ,.+10s/^/\t` um die nächsten 10 Zeilen einzurücken kann durch die Folge `qqI<Tab><Esc>jq` und `9@q` ersetzt werden. Ähnlich Beispiele:

Kommentar löschen: `qq0f%Djq`

Leerzeichen durch Tabulator ersetzen: `qq0d^i<Tab><Esc>jq`

Solche Zeilenmanipulationen starten häufig mit `0`, um einen Ausgangspunkt unabhängig der Cursorposition zu schaffen und enden mit `j`, um in die nächste Zeile zu wechseln, in welcher der Befehl erneut ausgeführt werden kann.

### ■ Wortmanipulationen Auch kleine wiederkehrende Arbeitsabläufe sollten in Macros gespeichert werden. `qkbi"<Esc>ei"<Esc>q` kann genutzt werden, um Wörter einzuklammern. $\text{\LaTeX}$ spezifische Anwendungen:

`qbi\textbf{<ESC>ei}<ESC>q`

oder `qqbi\cite{<Esc>ea}<Esc>jq`. `b` steht für back (zurück an den Wortanfang). Bei häufiger Nutzung evtl. Tastenfunktion in Konfigurationsdatei.

## ■ **L<sup>A</sup>T<sub>E</sub>X-spezifische Anwendungen**

- **Block in Umgebung einwickeln:**

`qq0\begin{Umgebung}Esc>5j0\end{Umgebung}<Esc>jq`

Ein Snippet wäre hier zielführender.

- **Nummerierung der Überschriften entfernen:** `qq0f\cw\section*<Esc>jq`

Springt zum Anfang der Zeile 0, sucht erstes \ und ändert Wort zu \section\*

- **Wechsel von itemize zu enumerate Umgebung:**

`qq/itemize<Enter>cwenumerate<Esc>yiwnp<Esc>q`

oder `qqcwenumerate<Esc>q` in Kombination mit /itemize

(yank-in-word wird gefolgt von next und paste)

- **\$\$ expandieren zu equation-Umgebung:**

`qq0f$ct\begin{equation}<Esc>A\end{equation}<Esc>jq`

A steht für append (in nächster Zeile).

# buffer

In-Memory-Repräsentation einer Datei oder Textinhalts.

**:help buffer**

- Beinhaltet die Rohdaten einer Datei im Arbeitsspeicher und einige Metadaten.  
(ID, Dateiname, Status, Typ, Flags)
- Mit **:w** werden Daten auf Datei geschrieben.
- Es können mehrere Buffer einer Datei geöffnet werden
- Nicht an ihre visuelle Darstellung gebunden (ein Buffer kann in mehreren (oder keinem) Fenstern/Tabs angezeigt werden).

**:new** neuer Buffer (blank)

**:e <Datei>** neuer Buffer von bestimmter Datei

**:buffers** listet Buffer

**:bn** **:bp** nächster oder vorheriger Buffer

**:bd** schließt Buffer (+! für forced)

# Anwendungsbeispiele



# Bericht

Ausgangslage für einen Laborbericht seien eine Reihe an Notizen und Bildern:

apfel.txt apfel.png birne.txt birne.png ...

## Beispiel Teil 1

- Laden der Vorlage und Öffnen des Editors:  
`$ cat TEMPLATES/latex/bericht1.tex » bericht.tex && nvim bericht.tex`
- Erfassen der Dateinamen in Speicherregister a:  
`:let @a=system("ls *.txt")`
- Einfügen in Dokument (vor `\end{document}`):  
`/doc<Enter>nn0<Esc>"ap`



- Kapitelvorlage mit Snippet erstellen und in Register t ablegen:  
`0\sectionÖÖÖ<Enter>figure<Tab> ... ÖÖÖ.jpg<Tab> ... <Tab><Esc>9k"t9dd`
- Start der Macroaufnahme und Einfügen von Notizen in Vorlage:  
`qq"nD"tp :e <C-r>n<Enter>ggVGy:bd<Enter> p`  
Das Macro startet, kopiert (Register n) und löscht den Zeileninhalt (apfel.txt). Nach der Einfügung der Kapitelvorlage aus Register t wird ein neuer Buffer mit dem zuvor gespeicherten Dateinamen (Register n) geöffnet. Dessen Inhalt wird in das Standardregister kopiert und im vorherigen Buffer eingefügt.
- Kapitelname in Vorlage substituieren und Aufzeichnung beenden:  
`:%s/ÖÖÖ/<C-r>n<Return(x4)>/g<Enter> /\.txt<Enter>:noh<Enter>0q`  
Der Befehl :noh entfernt die Markierungen der Vorwärtssuche.
- Wiederholen für sämtliche Notizen:  
`4@q`

# CSV

Daten einer CSV-Datei sind in einer Tabelle darzustellen:

## Beispiel

- Snippet aktivieren und Daten einfügen:  
`obegin<Tab>tabular<Esc>la{ |c<Esc>hyWl6pa| }<Tab>  
:e ex.csv<Enter>ggVGybdp`
- Macro aufnehmen:  
`qq0:s/,/ & /g<Enter>A\\<Esc>jq`
- Macro aufrufen:  
`10@q`



**JOHANNES KEPLER  
UNIVERSITY LINZ**