

Patchen ist easy



Ulrike Fischer, Bonn
L^AT_EX Project Team

24.6.2022
Magdeburg

Was ist Patchen?

- Änderung von *fremden* Befehlen
- Oft Änderung von *internen* Befehlen
- mit dem Ziel Funktionserweiterung oder -änderungen oder Fehlerkorrektur
- Völlige Umdefinition ist kein Patchen!



- ein Dokument patcht `\section`

```
\AddToHook{cmd/section/before}{\clearpage}
```

- `xcolor` patcht einen Befehl von `colortbl`:

```
% This is a fix for active `!'  
character to enable color expressions ...
```

```
\def\CT@extract#1\columncolor#2#3\@nil  
{\if!#2%  
...}
```

- `colortbl` patcht `longtable`

```
\def\LT@@hline{%  
  \ifx\LT@next\hline
```



- hyperref patcht alles mögliche

```
\long\def\@caption#1...
\let\float@makebox\HyNew@float@makebox
\def\@lbibitem[#1]#2{%
\def\contentsline#1#2#3#4
\def\@newctr#1[#2]{%
\def\@definecounter#1{%
\def\@chapter{%
\def\@sect#1#2#3#4#5#6[#7]#8{%
```

- hyperxmp patch kvoptions (fehlerhaft!)

```
\let\hyxmp@ProcessKeyvalOptions=\ProcessKeyvalOptions
\renewcommand*{\ProcessKeyvalOptions}{%
```



- Er geht überhaupt nicht.
- Subtile Verhaltensänderung.

```
\newcommand\foo{1}  
\setcounter{section}{\foo}
```

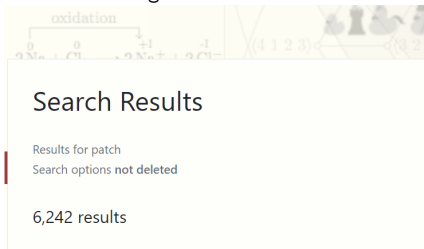
```
\renewcommand\foo{\textbf{1}}  
\setcounter{section}{\foo} %ups  
! Missing number, treated as zero.
```



Patchen tun andere auch!

follower.png

<https://tex.stackexchange.com/search?tab=relevance&q=patch>



The screenshot shows a search results page for the query 'patch'. At the top, there is a header with the word 'oxidation' and some chemical diagrams. Below that, the text 'Search Results' is prominently displayed. Underneath, it says 'Results for patch' and 'Search options not deleted'. At the bottom of the visible section, it indicates '6,242 results'. The background of the screenshot is slightly faded, showing various technical diagrams and text.



Worauf soll man achten?

- Welche Argumente hat der Befehl?
- Ist er global oder lokal definiert?
- Wann wird er definiert?
- Ist er expandierbar oder robust oder protected?
- Unter welchem Catcoderegime wurde der Befehl definiert?
- Wo wird er verwendet?
- Gibt es weitere Patches?



- Einfacher Befehl ohne Argumente

```
\def\mytestA{Hello}
```

```
> \mytestA=macro:  
->Hello.
```

```
\newcommand\mytestB{Hello}
```

```
> \mytestB=\long macro:  
->Hello.
```

- Einfacher Befehl mit Argumenten

```
\newcommand\mytestC[2]{Hello #1, #2!}
```

```
> \mytestC=\long macro:  
#1#2->Hello #1, #2!.
```



- „protected“ Befehle

```
\protected\def\mytestA{Hello}
```

```
> \mytestD=\protected macro:  
#1#2->Hello #1, #2!.
```

```
\show ä
```

```
> Ã=\protected macro:  
->\UTFviii@two@octets Ã.
```



- Befehl mit optionalem Argument: mit innerer Struktur:

```
\newcommand\mytestE[2][Welt]{Hello #1, #2!}
```

```
> \mytestE=macro:
```

```
->\@protected@testopt \mytestE \mytestE {Welt}.
```

- Robuste Befehle: mit innerere Struktur

```
\DeclareRobustCommand\mytestF{abc}
```

```
> \mytestF=macro:
```

```
->\protect \mytestF□.
```



- „xparse“-Befehle: mit innerer Struktur und protected

```
\NewDocumentCommand\mytestG{}{abc}
```

```
> \mytestG=\protected macro:
```

```
->\__cmd_start_expandable:nNNNNn {} \mytestG_␣  
    \mytestG_␣ \mytestG_␣code_␣?{}.
```

- komische Dinge

```
\show\sim
```

```
> \sim=\mathchar"3218.
```



Richtiges Catcoderegime verwenden!

- `\makeatletter`
- `\ExplSyntaxOn/\makeatletter`

```
\DeclareRobustCommand*\markboth[2]{....  
  \unrestored@protected@xdef\@themark  
    {{#1}{#2}}%  
  ....  
  \tl_if_empty:nF{#2}{  
    \mark_insert:nn{2e-right-nonempty}{#2} }  
  ...}
```

- Spezielles (colorspace):

```
\catcode`\&=11 \makeatletter  
\def\color@&spot#1#2{...}  
\catcode`\&=4 \makeatother
```



Patchmethoden: Umdefinition

```
\renewcommand\Vr@f [2] [] {%  
  \begingroup  
  \let\T@pageref \@pagerefstar  
  \hyperref [{#2}] {%  
    \Ref*{#2}  
    \vpageref [#1] {#2}%  
  }%  
 \endgroup  
}%
```



- Man bekommt exakt das, was man will.
- Die letzte (Um)definition gewinnt.
- Änderungen am Original werden nicht bemerkt

⇒ Check einbauen:

```
\newcommand\foo[1]{Hallo Welt #1!}  
\CheckCommand\foo[1]{hallo welt #1!}
```

```
LaTeX Warning: Command \foo has changed.  
Check if current package is valid.
```



```
\let\H@old@spart\@spart
```

```
\def\@spart#1{%  
  \Hy@MakeCurrentHrefAuto{part*}%  
  \Hy@raisedlink{%  
    \hyper@anchorstart{\@currentHref}\hyper@anchorend  
  }%  
  \H@old@spart{#1}%  
}
```



- Reagiert auf Änderungen der Originalbefehle
- Patches kumulieren:

```
\let\NR@spart\@spart
```

```
\long\def\@spart#1{%  
  \NR@getttitle{#1}%  
  \NR@spart{#1}%  
}
```



Patchmethoden: Let-Verfahren: Probleme

- Innere Struktur wird nicht kopiert:

```
\DeclareRobustCommand\mytest{abc}  
\let\orimytest\mytest
```

```
> \orimytest=macro:  
->\protect \mytest .
```

```
\DeclareRobustCommand\mytest{patch \orimytest}
```

```
\mytest -->  
\mytest_□ -->  
patch \orimytest -->  
\mytest_□ -->  
patch \orimytest -->
```



```
! TeX capacity exceeded, sorry [main memory  
size=5000000].
```

Let-Verfahren: innere Struktur kopieren

```
\LetLtxMacro\orimyttest\myttest % letltxmacro -Paket
```

```
%oder
```

```
\NewCommandCopy\orimyttest \myttest % aktuelles LaTeX
```

```
> \orimyttest=macro:
```

```
->\protect \orimyttest .
```



Patchmethoden: Vorne oder hinten anhängen

- `\g@addto@macro` L^AT_EX
`\appto \preto \gappto \gpreto` etoolbox
`\addto` babel
`\tl_put_left:Nn \tl_put_right:Nn` expl3
...

- Nur für einfache Befehle ohne Argumente
- Beispiel

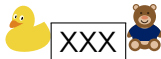
```
\appto\thesection{XXX}
```

```
> \thesection=macro:
```

```
->\@arabic \c@section XXX.
```



```
\AddToHook{cmd/fbox/before}{\tikz[scale=.3]{\duck}}  
\AddToHook{cmd/fbox/after}{\tikz[scale=.3]{\bear\bearwear}}  
\fbox{XXX}
```



- Code in Hooks kann sortiert und auch wieder entfernt werden.
- Funktioniert mit (fast) jedem Befehl
- Der Befehl kann Argumente haben
- Die Argumente können aber *nicht* im Hook verwendet werden.
- Vorne anhängen ist sicherer als hinten.



Anhängen mit etoolbox und xpatch

- Argumente können verwendet werden
- nicht geeignet für alle Befehlstypen
- etoolbox für einfachere Befehle

```
\newcommand\mytest[1]{a#1b}  
\pretocmd\mytest{hallo[#1]}{\}\{\fail}  
\mytest{XXX}
```

hallo[XXX]aXXXb

- xpatch für robuste Befehle

```
\DeclareRobustCommand\mytestB[1]{a#1b}  
\xpretocmd\mytestB{hallo[#1]}{\}\{\fail}  
\mytestB{XXX}
```



```
\newcommand\mytest[1]{Ente, Eule, Eule, #1, Kuh}  
\mytest{Hippo}  
\patchcmd\mytest{Eule}{Bär}{}{\fail} %oder \xpatchcmd  
\mytest{Wolf}
```

Ente, Eule, Eule, Hippo, Kuh
Ente, Bär, Eule, Wolf, Kuh

```
\patchcmd\mytest{Eule}{XXXX}{}{\fail}  
\patchcmd\mytest{Eule}{Bär}{}{\fail}  
\patchcmd\mytest{XXXX}{Eule}{}{\fail}
```

Ente, Eule, Bär, Hippo, Kuh



Patchmethoden: tracing

```
\tracingpatches  
\makeatletter  
\xpatchcmd\markboth{\begingroup}{\begingroup\let\cite\relax}{}{\fail}  
\makeatother
```

```
[debug] tracing \patchcmd on input line 66  
[debug] analyzing '\markboth '  
[debug] ++ control sequence is defined  
[debug] ++ control sequence is a macro  
[debug] -- macro cannot be retokenized cleanly  
[debug] -> the macro may have been defined under a category  
[debug] code regime different from the current one  
[debug] -> the replacement text may contain special control  
[debug] sequence tokens formed with \csname...\endcsname;  
[debug] -> the replacement text may contain carriage return,  
[debug] newline, or similar characters
```



Alles Lüge? Patchen ist doch nicht easy

- Patchen benötigt $\text{T}_{\text{E}}\text{X}$ und $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ -Programmierkenntnisse,
- Patchen benötigt Kenntnisse, wo und wie Befehle verwendet werden.
- Patchen kann Dokumente und Pakete kaputt machen.
- Patchen benötigt Kenntnisse, wie man Patches koordiniert.



- Patchen kann man schnell selbst machen.
- Man muss nicht auf Paketautoren warten.
- Patches müssen nicht perfekt sein.
- Patchen ist fein ... aber nur in Maßen!



Patchen vermeiden und Patches zurückdrehen ist viel mühseliger!

