<DirectoryMatch *Muster*> ... </DirectoryMatch>
Eingebettete Direktiven anwenden, wenn das Dateisystemverzeichnis *Muster* entspricht.

<FilesMatch *Muster*> ... </FilesMatch>
Eingebettete Direktiven anwenden, wenn Datei *Muster* entspricht.

<LocationMatch *Muster*> ... </LocationMatch>
Eingebettete Direktiven anwenden, wenn URL *Muster* entspricht.

<ProxyMatch *Muster*> ... </ProxyMatch>
Eingebettete Direktiven anwenden, wenn URL *Muster* entspricht.

Beispiele

Beispiel 31: Einfaches Matching

```
# /foo zu /bar umschreiben
RewriteEngine On
RewriteRule ^/foo$ /bar
```

Beispiel 32: Matching- und Capture-Gruppen

```
# Schöne URL zu Skriptparametern umschreiben
RewriteRule ^/(\w+)/(\d+) /index.php?action=$1&id=$2
```

Beispiel 33: Rewrite-Conditions

```
# Admin-URL auf interne IP-Adressen beschränken
RewriteCond %{REMOTE_ADDR} !192.168.\d*.\d*
RewriteCond %{PATH_INFO}  ^admin
RewriteRule .* - [F]
```

Beispiel 34: Auf SSL umleiten

```
# Sicherstellen, dass Admin-URLs über SSL geliefert werden
RewriteCond   %{SERVER_PORT}  !^443$
RewriteRule ^/admin/(.*)$ https://www.example.com/admin/$1 [L,R]
```


Der Editor vi

vi ist ein auf allen Unix-Systemen weit verbreiteter Texteditor, und *Vim* ist ein populärer *vi*-Klon mit einer erweiterten Unterstützung regulärer Ausdrücke. Beide verwenden eine DFA-Engine. Eine Erläuterung der Regeln, nach denen eine DFA-Engine arbeitet, finden Sie im Abschnitt »Einführung in reguläre Ausdrücke und in das Pattern-Matching«.

Unterstützte Metazeichen

Die Tabellen 56 bis 60 führen die von *vi* unterstützten Metazeichen und Metasequenzen auf. Ausführlichere Definitionen zu den einzelnen Metazeichen finden Sie weiter oben im Abschnitt »Regex-Metazeichen, Modi und Konstrukte«.

Tabelle 56: Darstellung von Zeichen

Sequenz	Bedeutung
Nur bei Vim	
\b	Backspace, \x08
\e	ESC-Zeichen, \x1B
\n	Newline, \x0A
\r	Carriage Return, \x0D
\t	Tabulator, \x09

Tabelle 57: Zeichenklassen und klassenartige Konstrukte bei vim

Klasse	Bedeutung
[...]	irgendein Zeichen aus der Liste
[^...]	irgendein Zeichen, das sich nicht in der Liste befindet
[:*Klasse*:]	Zeichenklasse im POSIX-Stil, nur innerhalb einer Zeichenklasse gültig
.	beliebiges Zeichen außer Newline (außer im /s-Modus)

Tabelle 57: Zeichenklassen und klassenartige Konstrukte bei vim (Fortsetzung)

Klasse	Bedeutung
Nur bei Vim	
`\w`	Wortzeichen, `[a-zA-Z0-9_]`
`\W`	Nicht-Wortzeichen, `[^a-zA-Z0-9_]`
`\a`	Buchstabe, `[a-zA-Z]`
`\A`	Nicht-Buchstabe, `[^a-zA-Z]`
`\h`	Anfang eines Wortzeichens, `[a-zA-Z_]`
`\H`	Nicht-Anfang eines Wortzeichens, `[^a-zA-Z_]`
`\d`	Ziffer, `[0-9]`
`\D`	Nicht-Ziffer, `[^0-9]`
`\s`	Whitespace-Zeichen, `[ \t]`
`\S`	Nicht-Whitespace-Zeichen, `[^ \t]`
`\x`	Hex-Ziffer, `[a-fA-F0-9]`
`\X`	Nicht-Hex-Ziffer, `[^a-fA-F0-9]`
`\o`	oktale Ziffer, `[0-7]`
`\O`	nicht-oktale Ziffer, `[^0-7]`
`\l`	Kleinbuchstabe, `[a-z]`
`\L`	Nicht-Kleinbuchstabe, `[^a-z]`
`\u`	Großbuchstabe, `[A-Z]`
`\U`	Nicht-Großbuchstabe, `[^A-Z]`
`\i`	über `isident` definiertes Bezeichnerzeichen
`\I`	wie `\i`, aber Ziffern ausschließend
`\k`	über `iskeyword` definiertes Schlüsselwortzeichen, häufig durch Sprachmodi gesetzt
`\K`	wie `\k`, aber Ziffern ausschließend
`\f`	über `isfname` definiertes Dateinamenzeichen; betriebssystemabhängig
`\F`	wie `\f`, aber Ziffern ausschließend
`\p`	über `isprint` definierte druckbare Zeichen, üblicherweise `x20-x7E`
`\P`	wie `\p`, aber Ziffern ausschließend

Tabelle 58: Anker und Zusicherungen der Länge null bei vi

Sequenz	Bedeutung
`^`	Anfang einer Zeile, wenn es in einer Regex an erster Stelle steht, anderenfalls steht es für sich selbst
`$`	Ende einer Zeile, wenn es in einer Regex am Ende steht, anderenfalls steht es für sich selbst
`\<`	Anfang der Wortgrenze (d.h. die Position zwischen einem Interpunktions- oder Leerzeichen und einem Wortzeichen)
`\>`	Ende der Wortgrenze

Tabelle 59: Modus-Modifikator bei vim

Modifikator	Bedeutung
`:set ic`	Ignoriert die Groß-/Kleinschreibung bei allen Suchen und Substitutionen.
`:set noic`	Beachtet die Groß-/Kleinschreibung.
`\u`	Nächstes Zeichen im Ersetzungs-String wird großgeschrieben.
`\l`	Nächstes Zeichen im Ersetzungs-String wird kleingeschrieben.
`\U`	Nachfolgende Zeichen im Ersetzungs-String werden großgeschrieben.
`\L`	Nachfolgende Zeichen im Ersetzungs-String werden kleingeschrieben.
`\E` oder `\e`	Beendet den mit `\U` oder `\L` eingeleiteten Bereich.

Tabelle 60: Gruppierung, Capturing, Conditionals und Kontrolle

Sequenz	Bedeutung
`\(...\)`	Submuster gruppieren und Subtreffer in `\1`, `\2`, ... ablegen
`\n`	enthält das Ergebnis des *n*-ten vorangegangenen Subtreffers; sowohl in Regex-Mustern als auch in Ersetzungs-Strings gültig
`&`	evaluiert in einem Ersetzungs-String zum erkannten Text
`*`	keinmal oder mehrmals
Nur bei Vim	
`\+`	einmal oder mehrmals
`\=`	einmal oder keinmal
`\{n}`	genau *n*-mal
`\{n,}`	mindestens *n*-mal

Tabelle 60: Gruppierung, Capturing, Conditionals und Kontrolle (Fortsetzung)

Sequenz	Bedeutung
\{,n}	höchstens *n*-mal
\{x,y}	mindestens *x*-mal, aber nicht öfter als *y*-mal

Pattern-Matching

Suche

/Muster
?Muster

Bewegt sich zum Beginn der nächsten Position, an der das *Muster* in der Datei erkannt wurde. Ein *?Muster* sucht rückwärts. Eine Suche kann mit n (vorwärts) oder N (rückwärts) wiederholt werden.

Substitution

:[*Adresse1*[,*Adresse2*]]s/*Muster*/*Ersatz*/[cgp]

Ersetzt in jeder Zeile des Adressbereichs den durch *Muster* erkannten Text durch *Ersatz*. Ohne Angabe eines Adressbereichs wird die aktuelle Zeile verwendet. Jede Adresse kann eine Zeilennummer oder ein regulärer Ausdruck sein. Wird *Adresse1* übergeben, beginnt die Substitution an dieser Zeilennummer (bzw. an der ersten passenden Zeile) und erstreckt sich bis zum Dateiende oder bis zu der durch *Adresse2* festgelegten Zeile. Es gibt auch eine Reihe von Adresskürzeln, die in den folgenden Tabellen beschrieben werden.

Substitutionsoptionen

Option	Bedeutung
c	bei jeder Substitution rückfragen
g	alle Treffer einer Zeile ersetzen
p	Zeile nach der Substitution ausgeben

Adresskürzel

Adresse	Bedeutung
.	aktuelle Zeile
$	letzte Zeile der Datei
%	gesamte Datei
't	Position »t«
/...[/]	nächste vom Muster erkannte Zeile
?...[?]	vorherige vom Muster erkannte Zeile
\/	nächste von der letzten Suche erkannte Zeile
\?	vorherige von der letzten Suche erkannte Zeile
\&	nächste Zeile, bei der das letzte Substitutionsmuster zutrifft

Beispiele

Beispiel 35: Einfache Suche bei vi

```
Finde spider-man, Spider-Man, Spider Man
/[Ss]pider[- ][Mm]an
```

Beispiel 36: Einfache Suche bei Vim

```
Finde spider-man, Spider-Man, Spider Man, spiderman, SPIDER-MAN usw.
:set ic
/spider[- ]\=man
```

Beispiel 37: Einfache Substitution bei vi

```
Global <br> XHTML-konform in <br /> umwandeln
:set ic
: % s/<br>/<br \/>/g
```

Beispiel 38: Einfache Substitution bei Vim

```
Global <br> XHTML-konform in <br /> umwandeln
: % s/<br>/<br \/>/ig
```

Beispiel 39: Komplexere Substitution bei Vim

```
Urlify - URLs in HTML-Links umwandeln
: % s/\(https\=:\/\/[a-z_.\\w\/\\#~:?+=&;%@!-]*\)/< a href=
"\1">\1<\/a>/ic
```

Zusätzliche Ressourcen

- *Learning the vi Editor*, 6. Auflage, von Linda Lamb und Arnold Robbins (O'Reilly), ist eine Einführung in den *vi*-Editor und populäre *vi*-Klone.
- *http://www.geocities.com/volontir/* von Oleg Raisky bietet eine Übersicht zur Regex-Syntax von *Vim*.

Shell-Tools

awk, *sed* und *egrep* bilden eine Gruppe von Unix-Shell-Tools zur Bearbeitung von Text. *awk* verwendet eine DFA-Engine, *egrep* schaltet, je nachdem, welche Features verwendet werden, zwischen einer DFA- und einer NFA-Engine um, und *sed* arbeitet mit einer NFA-Engine. Eine Erläuterung der Regeln, nach denen sich diese Engines richten, finden Sie im Abschnitt »Einführung in reguläre Ausdrücke und in das Pattern-Matching«.

Diese Referenz behandelt GNU *egrep* 2.4.2 (ein Programm, das nach Textzeilen sucht), GNU *sed* 3.02 (ein Scripting-Tool für Editierbefehle) und GNU *awk* 3.1 (eine Programmiersprache zur Textbearbeitung).

Unterstützte Metazeichen

awk, *egrep* und *sed* unterstützen die in den Tabellen 61 bis 65 aufgeführten Metazeichen und Metasequenzen. Ausführlichere Definitionen zu den einzelnen Metazeichen finden Sie weiter oben im Abschnitt »Regex-Metazeichen, Modi und Konstrukte«.

Tabelle 61: Darstellung von Zeichen in Shell-Tools

Sequenz	Bedeutung	Tool
`\a`	Alarm (BEL)	*awk, sed*
`\b`	Backspace; wird nur in Zeichenklassen unterstützt	*awk*
`\f`	Formfeed	*awk, sed*
`\n`	Newline (Linefeed)	*awk, sed*
`\r`	Carriage Return	*awk, sed*
`\t`	Tabulator	*awk, sed*
`\v`	vertikaler Tabulator	*awk, sed*
`\ooktal`	über ein, zwei oder drei Oktalziffern spezifiziertes Zeichen	*sed*
`\oktal`	über ein, zwei oder drei Oktalziffern spezifiziertes Zeichen	*awk*
`\xhex`	über zwei Hexadezimalziffern spezifiziertes Zeichen	*awk, sed*
`\ddezimal`	über ein, zwei oder drei Dezimalziffern spezifiziertes Zeichen	*awk, sed*
`\cZeichen`	ein benanntes Kontrollzeichen (`\cC` ist z. B. Control-C)	*awk, sed*
`\b`	Backspace	*awk*
`\Metazeichen`	schützt das Metazeichen, sodass es sich selbst literal darstellt	*awk, sed, egrep*

Tabelle 62: Zeichenklassen und klassenartige Konstrukte in Shell-Tools

Sequenz	Bedeutung	Tool
`[...]`	erkennt ein beliebiges Zeichen aus der Liste	*awk, sed, egrep*
`[^...]`	erkennt ein beliebiges Zeichen, das sich nicht in der Liste befindet	*awk, sed, egrep*
`.`	erkennt ein beliebiges Zeichen außer Newline	*awk, sed, egrep*
`\w`	erkennt ein ASCII-Wortzeichen, `[a-zA-Z0-9_]`	*egrep, sed*
`\W`	erkennt ein Nicht-ASCII-Wortzeichen, `[^a-zA-Z0-9_]`	*egrep, sed*

Tabelle 62: Zeichenklassen und klassenartige Konstrukte in Shell-Tools (Fortsetzung)

Sequenz	Bedeutung	Tool
`[:Property:]`	erkennt ein Zeichen aus der POSIX-Zeichenklasse	*awk, sed*
`[^[:Property:]]`	erkennt ein Zeichen, das nicht in der POSIX-Zeichenklasse ist	*awk, sed*

Tabelle 63: Anker und andere Testshell-Tools für Zusicherungen der Länge null in Shell-Tools

Sequenz	Bedeutung	Tool
^	erkennt nur den Anfang des Strings, selbst wenn Newlines eingebettet sind	*awk, sed, egrep*
$	erkennt nur das Ende des Strings, selbst wenn Newlines eingebettet sind	*awk, sed, egrep*
\<	erkennt den Anfang einer Wortgrenze	*egrep*
\>	erkennt das Ende einer Wortgrenze	*egrep*

Tabelle 64: Kommentare und Modus-Modifikatoren in Shell-Tools

Modifikator	Bedeutung	Tool
Flag: `i` oder `I`	Groß-/Kleinschreibung ignorierendes Matching für ASCII-Zeichen	*sed*
Kommandozeilenoption: `-i`	Groß-/Kleinschreibung ignorierendes Matching für ASCII-Zeichen	*egrep*
`IGNORECASE` auf Wert ungleich `null` setzen	Groß-/Kleinschreibung ignorierendes Matching für Unicode-Zeichen	*awk*

Tabelle 65: Gruppierung, Capturing, Conditionals und Kontrolle in Shell-Tools

Klasse	Bedeutung	Tool
(*MUSTER*)	Gruppierung	*awk*
\(*MUSTER*\)	Gruppieren und Subtreffer in \1, \2, ..., \9 festhalten	*sed*
n	enthält den *n*-ten vorherigen Subtreffer	*sed*
...|...	Alternation; eine der Alternativen erkennen	*egrep, awk, sed*

Tabelle 65: Gruppierung, Capturing, Conditionals und Kontrolle in Shell-Tools (Fortsetzung)

Klasse	Bedeutung	Tool
Gierige Quantoren		
*	keinmal oder mehrmals	*awk, sed, egrep*
+	einmal oder mehrmals	*awk, sed, egrep*
?	einmal oder keinmal	*awk, sed, egrep*
\{*n*\}	genau *n*-mal	*sed, egrep*
\{*n*,\}	mindestens *n*-mal	*sed, egrep*
\{*x*,*y*\}	mindestens *x*-mal, aber nicht öfter als *y*-mal	*sed, egrep*

egrep

`egrep [Optionen] Muster Dateien`

egrep durchsucht die *Dateien* auf Vorkommen von *Muster* und gibt die entsprechenden Zeilen aus.

Beispiel

```
$ echo 'Spiderman Menaces City!' > dailybugle.txt
$ egrep -i 'spider[- ]?man' dailybugle.txt
Spiderman Menaces City!
```

sed

```
sed '[Adresse1][,Adresse2]s/Muster/Ersatz/[Flags]' Dateien
sed -f Skript Dateien
```

Standardmäßig wendet *sed* die Substitution auf alle Zeilen in den *Dateien* an. Jede Adresse kann entweder eine Zeilennummer oder ein Regex-Muster sein. Ein angegebener regulärer Ausdruck muss zwischen Slashes (`/...`) definiert sein.

Wurde *Adresse1* angegeben, beginnt die Substitution bei dieser Zeilennummer (bzw. bei der ersten zutreffenden Zeile) und erstreckt sich entweder bis zum Ende der Datei oder bis zu der durch *Adresse2* angegebenen Zeile. Die zwei Subsequenzen & und *n* werden in *Ersatz*, basierend auf dem Ergebnis des Matching interpretiert.

Die Sequenz & wird durch den durch *Muster* erkannten Text ersetzt. Die Sequenz *n* entspricht einer Capture-Gruppe (1...9) des aktuellen Treffers. Die folgenden Flags stehen zur Verfügung:

n
: Ersetzt den *n*-ten Treffer einer Zeile, wobei *n* zwischen 1 und 512 liegen kann.

g
: Ersetzt alle Vorkommen von *Muster* in einer Zeile.

p
: Gibt Zeilen mit erfolgreichen Substitutionen aus.

w *Datei*
: Schreibt Zeilen mit erfolgreichen Substitutionen in die *Datei*.

Beispiel

Wandelt Datumsangaben im Format MM/TT/JJJJ nach TT.MM.JJJJ um.

```
$ echo 12/30/1969' |
 sed 's!\([0-9][0-9]\)/\([0-9][0-9]\)/\([0-9]\{2,4\}\)!\2.\1.
\3!g'
```

awk

```
awk 'Anweisungen' Dateien
awk -f Skript Dateien
```

Das *awk*-Skript in *Anweisungen* oder *Skript* muss aus einer Folge von */Muster/ {Aktion}*-Paaren bestehen. Der *Aktion*-Code wird auf jede Zeile angewandt, die durch das *Muster* erkannt wurde. *awk* besitzt auch verschiedene Funktionen zur Mustererkennung.

Funktionen

match(*Text*, *Muster*)
: Ist das *Muster* im *Text* enthalten, wird die Position im *Text* zurückgegeben, an der der Treffer beginnt. Ein fehlgeschlagenes Matching gibt null zurück. Ein erfolgreiches Matching setzt auch die Variable RSTART auf die Anfangsposition des Treffers, und die Variable RLENGTH enthält die Anzahl der Zeichen im Treffer.

gsub(*Muster*, *Ersatz*, *Text*)
: Ersetzt alle Vorkommen von *Muster* im *Text* durch *Ersatz* und gibt die Anzahl der Substitutionen zurück. Nutzt standardmäßig $0, wenn kein *Text* angegeben wird.

sub(*Muster*, *Ersatz*, *Text*)
: Ersetzt das erste Vorkommen von *Muster* im *Text* durch *Ersatz*. Eine erfolgreiche Substitution liefert 1 zurück, anderenfalls wird 0 zurückgegeben. Nutzt standardmäßig $0, wenn kein *Text* angegeben wird.

Beispiel

Legen Sie eine *awk*-Datei an und führen Sie sie anschließend über die Kommandozeile aus.

```
$ cat sub.awk
{
    gsub(/https?:\/\/[a-z_.\\w\/\\#~:?+=&;%@!-]*/,
                    "<a href=\"\&\">\&</a>");

    print
}

$ echo "Besuchen Sie die Website http://www.oreilly.de/catalog/
regexpprger/" | awk -f sub.awk
```

Zusätzliche Ressourcen

- *sed & awk* von Dale Dougherty und Arnold Robbins (O'Reilly) ist eine Einführung und Referenz für beide Tools.

2. AUFLAGE

Reguläre Ausdrücke

kurz & gut

Tony Stubblebine

Deutsche Übersetzung von
Peter Klicman & Lars Schulten

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo