Auszug aus dem Buch »Adobe InDesign CS3 - Das Nachschlagewerk für Fortgeschrittene«

von Gerald Singelmann

Erschienen bei Galileo-Press und erhältlich unter anderem bei Amazon.de

3 GREP und reguläre Ausdrücke

Vorwort

Ich komme in meinen InDesign-Schulungen immer wieder an den Punkt (zum Beispiel bei verschachtelten Formaten), wo ein Grafiker den Kopf schüttelt und fragt: »Wer denkt sich denn so was aus?«. Und ich muss dann antworten »Informatiker und Programmierer«. Es ist unbestritten, dass Informatiker in mancher Hinsicht anders sind. Ich bin selbst einer, ich sollte es wissen. Es ist ebenso unbestritten, dass (wie der Name schon sagt) Informatiker Wege kennen, präzise mit Information und Daten umzugehen.

Reguläre Ausdrücke sind das Paradebeispiel. Auf den ersten Blick wirken die so, als habe ein wahnsinniger Zen-Mönch mit Kopfschmerzen auf eine Tastatur eingeschlagen. Auf den zweiten Blick kann ich Vorgänge in 20 Buchstaben beschreiben oder auslösen, die sonst eine Beschreibung von fünf Absätzen bräuchten und dann immer noch missverständlich wären.

Was ist GREP?

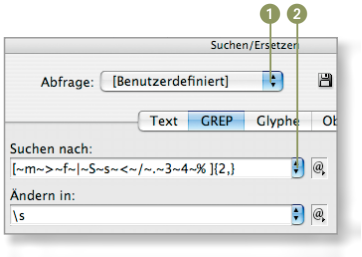
Ganz grob gesagt: GREP verhält sich zu einer normalen Suchen/Ersetzen-Funktion so wie James Bonds Aston Martin zu einem alten Opel Kadett. Auch eine alte Klapperkiste bringt mich von hier nach da, aber nur, wenn nicht die Kasseler Berge dazwischen liegen oder Blofelds Leute hinter mir her sind.

Quasi alle Suchen/Ersetzen-Anfragen in Diskussionsforen der letzten Jahre ließen sich mit GREP erledigen, auch wenn die ›alte‹ Form das nur umständlich oder gar nicht konnte.

GREP hat einfach alles Sonderzubehör inklusive Schleudersitz, der durch einen falsch eingegebenen Buchstaben ausgelöst wird.

Buchstäbliches finden

Dies funktioniert genauso wie beim Suchen/Ersetzen, das Sie kennen: Katze sucht nach genau diesem Wort und findet Katze. Es gibt allerdings eine Reihe von Zeichen, die eine besondere Bedeutung haben. So findet z. B. Was? nicht Was?, sondern Was oder Wa.



▲ Abbildung 3.53
In CS3 können Sie Suchen speichern 1. Die Suchanfragen seit Start des Programmes stehen in 2.

Auszeichnung der Suchbegriffe

In diesem Abschnitt sehen Sie viele rot und grün unterstrichene Begriffe. Ich brauchte eine einfache und deutliche Notation für die Begriffe, die Sie in das Suchfeld eintragen und die Zeichenketten, die damit gefunden werden.

Achten Sie darauf, dass dies und dies nicht identisch sind. Letzteres enthält auch das Leerzeichen.

Reservierte Zeichen

.	?	(	)
[	\	^	$
|	*	*	

Soll eines davon buchstäblich gefunden werden, setzen Sie ein \ davor. \. findet also wirklich einen Punkt.

Variationen finden

Es gibt zwei Ansätze, Variationen zu beschreiben: Zeichen für Zeichen oder Wort für Wort.

Zeichen für Zeichen

Angenommen, im Text ist von einer Sylke bzw. einer Silke die Rede, Sie wissen aber nicht genau, wie die Dame geschrieben wird. Suchen Sie einfach nach S[yi]lke. Jeder Buchstabe in den eckigen Klammern gilt.

Sie können in die Klammern auch Bereiche eingeben. Wenn Sie alle Automodelle von Audi suchen, geben Sie entweder A[23456] oder A[2-6] ein. Das findet A2 und A6, aber nicht A7 oder A26.

Diese Syntax wird häufig gebraucht, wenn Sie mit Wörtern arbeiten, die nur bestimmte Zeichen enthalten dürfen. Angenommen, Sie bauen einen Katalog mit Bestellnummern, in denen nur die Ziffern und die Buchstaben von A bis F vorkommen. Das beschreibt [0-9A-F], oder falls die Groß-/Kleinschreibung keine Rolle spielt: [0-9a-fA-F].

Kürzel
Viele Zeichenmengen kommen so häufig vor, dass es Kürzel dafür gibt. Eine Ziffer zum Beispiel wäre [0-9], kann aber auch als \d geschrieben werden. Eine Übersicht gibt es am Ende des Kapitels.

Finetuning
A[2-6] findet zwar nicht A26, aber es findet A26. Wir müssten also eigentlich noch ausschließen, dass nach [2-6] eine weitere Ziffer folgt. Wie das geht, besprechen wir weiter unten.

Sie können auch nach Zeichen suchen, die **nicht** in einem bestimmten Bereich liegen. Audi A[^3-6] findet alle Vorkommnisse, in denen Audi A nicht von 3, 4, 5 oder 6 gefolgt wird.

In den Klammern sind nur **^ \ -** und **]** reserviert. [[.\]] findet also [oder] oder .

In Kürze
[abc] findet a, b oder c.
[^abc] findet jedes Zeichen außer a, b oder c.

In Kürze
a|b findet a oder b.
a(b|c) findet ab oder ac.

Wort für Wort

Wenn Automodelle einfach durchnummeriert werden wie bei Audi, ist die Suche relativ einfach. Aber was machen wir bei einem Ford? Da können wir nach Ka|Fiesta|Focus|Mondeo suchen. Falls wir die Alternativen nur im Kontext mit dem Wort davor brauchen, werden die Alternativen geklammert, also Ford (Ka|Fiesta|Focus|Mondeo). Das findet Ford Focus, aber nicht nur Focus.

Reguläre Ausdrücke sind »eifrig« (im Englischen: »The regex engine is eager«). Es wird der erste Treffer geliefert, der möglich ist, und andere werden dadurch gegebenenfalls übersehen. Es ist also nicht egal, ob Sie nach Max|Maximilian oder nach Maximilian|Max suchen. Der erste Suchbegriff wird niemals Maximilian finden, da er mit Max schon vollkommen zufrieden ist.

Was im Rechner passiert
Suche da|dort in »Wenn du da bist«.
Ist ›W‹=›d‹(a)? ist ›W‹=›d‹(ort)?
Ist ›e‹=›d‹? Ist ›e‹=›d‹?
Ist ›n‹=›d‹? Ist ›n‹=›d‹?
Ist ›n‹=›d‹? Ist ›n‹=›d‹?
Ist › ‹=›d‹? Ist › ‹=›d‹?
Ist ›d‹=›d‹(a)? **Ja.** Ist ›u‹=›a‹?
Nein, zurück.
Ist ›d‹=›d‹(ort)? **Ja.** Ist ›u‹=›o‹?
Nein, weiter.
Ist ›u‹=›d‹? Ist ›u‹=›d‹?
Ist › ‹=›d‹? Ist › ‹=›d‹?
Ist ›d‹=›d‹? **Ja.** Ist ›a‹=›a‹?
Ja, Fertig.
Er ist mit ›da‹ also zufrieden, bevor er über die Alternative nachdenkt.

Optionale Teile

Die Suche nach Max eben war »Entweder ›Maximilian‹ oder ›Max‹«. Wir könnten dieselbe Suche auch beschreiben als »›Max‹, eventuell gefolgt von ›imilian‹«.

Der reguläre Ausdruck dafür ist Max(imilian)?. Das nachgestellte Fragezeichen bedeutet, dass das Zeichen direkt davor optional ist. Th?orsten findet also sowohl Torsten als auch Thorsten.

Wiederholungen (konkrete Anzahl)

Nehmen wir noch einmal das Beispiel der Bestellnummern. Oben habe ich nur beschrieben, wie man das einzelne Zeichen in der Bestellnummer findet. Was machen wir mit der ganzen Nummer? Angenommen, die Regeln sind »Ein oder zwei Buchstaben zwischen A und F, gefolgt von 4, 5 oder 6 Ziffern«.

Das könnte man mit dem optionalen Zeichen ? ausdrücken: [A-F][A-F]?[0-9][0-9][0-9][0-9][0-9]?[0-9]?. Nicht sehr elegant. Besser gehts, wenn wir sagen, wie viele Ziffern da stehen dürfen: [A-F]{1,2}[0-9]{4,6} findet A1234, aber auch BA41414.

Eine sehr simple Möglichkeit, eine Telefonnummer zu finden: [0-9]{3,5}/[0-9]{4,8} beschreibt sowohl 030/43189810 als auch 04104/1234.

Wenn statt des / auch - erlaubt sein soll, sieht der Suchbegriff so aus: [0-9]{3,5}[/-][0-9]{4,8}.

Wiederholungen (beliebige Anzahl)

Wenn letztlich offen ist, wie viele Ziffern in einer Bestellnummer sind, können wir auch + oder * anstelle des Intervalls { } nehmen.

- + findet eines oder mehrere Zeichen des Suchbegriffs/Zeichens links vom +.
- * akzeptiert beliebig viele Wiederholungen des Suchbegriffs, auch keins.
- hm+ findet hm, hmmm und hmmmmmm, aber nicht h.
- hm* findet das gleiche wie hm+, aber auch h.

Meistens werden + und * zusammen mit Wildcards verwendet. Das kommt jetzt:

Wildcards

Wildcards bezeichnen irgendein Zeichen mit bestimmten Einschränkungen. Die allgemeinste Wildcard ist der Punkt . Mit dem Punkt ist jedes Zeichen gemeint, außer der Absatzschaltung. Das Ergebnis von . ist also auf einen Absatz beschränkt. Wollen Sie wirklich nach einem Punkt suchen (und nicht nach einem beliebigen Zeichen), geben Sie \. ein.

.+\. findet also einen ganzen Satz, der mit einem Punkt aufhört? Nein:

Gierig und eifrig

Warum ist Max(imilian)? nicht schon mit Max zufrieden? Weil GREP nicht nur eifrig, sondern auch gierig ist. Er findet den ersten Treffer, aber davon so viel wie möglich. Solange also noch ein Zeichen zu dem Suchmuster passt, das er grad am Wickel hat, hört er nicht auf.

Das | verbindet zwei alternative Suchmuster. Wenn das vordere gefunden wird, wird das hintere ignoriert.

Lesbarkeit

(?x) ignoriert Leerzeichen im Suchbegriff.
Statt [0-9]{3,5}[/-][0-9]{4,8} könnte ich also auch
(?x) [0-9] {3,5} [/-] [0-9] {4,8}
schreiben.

In Kürze

x?	kein oder 1
x*	kein oder so viel wie möglich
x+	ein oder so viel wie möglich
x+?	ein oder so wenig wie möglich
x{2}	zwei
x{2,4}	zwei, drei oder vier; so viel wie möglich
x{2,}	mindestens zwei oder so viel wie möglich
x{2,4}	zwei, drei oder vier, so wenig wie möglich

Vorsicht: Gierige Suchzeichen

+, * und {2,9} sind »gierig«. Sie finden die längste Zeichenkette, die der Beschreibung entspricht.

Nehmen Sie den HTML-Text »Dieser Satz enthält zwei fette Wörter«.

Der Suchbegriff <.+> findet gierigerweise Satz enthält zwei und nicht nur Satz.

.+\. findet demnach nicht nur einen Satz, sondern alles bis zum letzten Punkt im Absatz.

Es gibt »faule« Variationen von +, * und {n,m}, indem Sie ein Fragezeichen hinter das Zeichen setzen. .+?\. findet nur den ersten Satz im Absatz. .+? bedeutet so viel wie »Finde die kürzeste Entsprechung«. Das Problem kann sein, dass GREP hier umständlich, weil wirklich faul vorgeht. Wenn wir .+?\. auf diesen Absatz anwenden, passiert im Computer Folgendes:

- .+? heißt kürzeste Entsprechung, also reicht E
- Mal schauen, was kommt als Nächstes: \. Ist s ein Punkt?
- Nein, Mist, ich muss zurück.
- Vielleicht reicht ja Es für .+?.
- Ist _ ein Punkt?
- Nein, Mist, ich muss zurück.

Effizienter ist es in diesem Fall, die Suche umzuformulieren: »Alle Zeichen außer einem Punkt, die von einem Punkt gefolgt werden«: [^.]+\. Wenn Sie das probieren, stoßen Sie auf ein weiteres Problem: [^.] findet auch Absatzschaltungen. Falls es Absätze ohne . am Ende gibt, werden also auch mehrere Absätze gefunden. Besser müsste es heißen: [^.\n\r]+\. oder »Alle Zeichen außer einem Punkt, einer harten Zeilenschaltung oder einem Absatzende, die von einem Punkt gefolgt werden«.

Wildcards II – Reloaded ;)

Es gibt auch speziellere Wildcards. \d z. B. entspricht [0-9], bezeichnet also eine beliebige Ziffer. Alle Wildcards lassen sich Zeichenmenge [ ] beschreiben, wobei die Liste zwischen den eckigen Klammern sehr lang werden kann.

\w findet ein Zeichen, das in einem Wort vorkommen kann. Also keine Satzzeichen, Absatzschaltungen etc. Das entspricht im Deutschen in etwa [a-zA-ZäöüÄÖÜß]. InDesign nimmt natürlich die Sonderzeichen der anderen Sprachen mit die Sammlung auf.

\W findet das Gegenteil, also alle Zeichen, die nicht Teil eines Wortes sind, in InDesign allerdings auch den Divis -.

\s ist ein Weißraum (*space character*), also alles, was im Menü Schrift unter Leerzeichen einfügen steht, aber auch Tabs und Zeilenschaltungen.

In Kürze

\d	Ziffer
\D	keine Ziffer
\w	Wortbuchstabe
\W	kein Wortteil
\s	space, beliebiger Leerraum
\S	kein Leerraum
\l	lower case, Kleinbuchstabe
\L	kein Kleinbuchstabe
\u	upper case, Großbuchstabe
\U	kein Großbuchstabe

\S ist das Gegenteil, also alle Zeichen, die kein Leerzeichen sind.

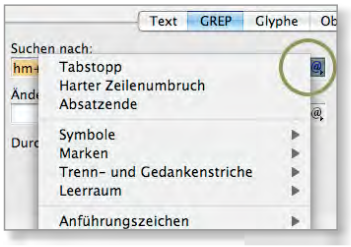
Sonderzeichen

Vielleicht wissen Sie aus InDesigns bisheriger Suchen/Ersetzen-Funktion, dass man auch nach Sonderzeichen wie zum Beispiel einem Tabulator suchen kann. In der klassischen Suche geben Sie diese Zeichen mit einem Caret ein. ^t ist dort das Zeichen für einen Tabulator. GREP hat eine Reihe standardisierter Sonderzeichen, die per Schrägstrich gekennzeichnet werden:

- \t Tabulator (entspricht dem alten ^t)
- \n Harter Zeilenumbruch (entspricht dem alten ^n)
- \r Absatzschaltung (entspricht dem alten ^p)

InDesign hat eine Reihe eigener Sonderzeichen wie den Tabulator für Einzug rechts (Shift + ⇆, ^y). Diese werden für GREP in der Regel durch eine Tilde gekennzeichnet (da ^ schon reserviert ist), hier also ~y.

Von diesen InDesign-eigenen Sonderzeichen gibt es so viele, dass ich froh bin, die auch über den Knopf mit dem @ rechts neben Suchfeld eingeben zu können.



▲ Abbildung 3.54
Haufenweise Sonderzeichen finden Sie am Ende des Kapitels oder im Popup-Menü des Suchen-Dialogs.

Positionsfestlegung

Oft ist es wichtig, das erste oder letzte Zeichen in einem Wort oder Absatz zu finden. ^ steht für den Absatzanfang, $ für das Absatzende, \b für Wortanfang oder Wortende (je nachdem, wie es gebraucht wird). Beispiele:

- \bN\w+ findet alle Wörter, die mit einem N anfangen.
- \b\w+en\b findet alle Wörter, die mit en aufhören.
- \. $ findet das Ende von allen Absätzen, die nach dem letzten Punkt noch ein Leerzeichen haben. \.\s$ wäre noch allgemeiner, da es alle Leerzeichen berücksichtigt. (Beachten Sie, dass . $ irgendein Zeichen vor dem Leerzeichen vor dem Absatzende suchen würde.)
- ^Ich\s findet alle Absätze, die mit Ich anfangen. InDesign findet mit ^ auch die Stelle direkt nach einem harten Zeilenumbruch. Wenn es wirklich ein Absatzanfang sein soll, verwenden Sie einen *positive Lookbehind* (siehe übernächste Seite).

In Kürze

\b	Wortgrenze (border) vorne oder hinten
\<	Wortanfang
\>	Wortende
^	Absatzanfang
$	Absatzende
\A	Anfang Textfluss
\Z	Ende Textfluss, aber vor einer letzten Absatzschaltung
\z	Ende Textfluss, nach einer letzten Absatzschaltung

Speicher für Teilergebnisse

Wir haben bisher in manchen Suchtermini Klammern benutzt, um Teilbereiche abzufragen; zum Beispiel waren in der Suche nach dem Ford-Modell Ford (Focus|Ka) die Alternativen für die Modellbezeichnungen geklammert.

Die Klammern sorgen einerseits dafür, dass Ford auf jeden Fall

In Kürze

$1	Das Gefundene der ersten Klammer: Ersetzen-Feld
\1	Das Gefundene der ersten Klammer im Suchen-Feld (für Wiederholungen)
(?)	Gruppe, die nicht gespeichert wird

vor dem Modell stehen muss. Andererseits merkt sich GREP das Teilergebnis der Klammer in einer Variablen $1. (Wenn mehrere Klammern vorkommen, wird die Zahl nach $ hochgezählt bis 9.) Wenn Sie den Text »Auf dem Parkplatz stand ein Ford Ka« mit (Ford (Ka|Focus)) durchsuchen, steht in $1 Ford Ka und in $2 Ka, da die äußere Klammer zuerst kommt.

Was soll das? Zum einen ist diese Speicherung in $n sehr praktisch zum Ersetzen. Ich schreibe dieses Kapitel nicht direkt in InDesign und muss die Suchterme kennzeichnen, damit sie später im Layout entsprechend formatiert werden können. Dazu schließe ich Suchterme in ein besonderes Zeichen ein, das sonst im Text nicht vorkommt, schreibe also etwa »#^Ich\s#«. Dann führe ich in InDesign eine Suche nach #([^#]+)# durch und ersetze durch $1, wobei ich dem auch noch ein Zeichenformat zuweisen kann. So wird mit einem Schlag alles formatiert und alle »#« fliegen raus, da $1 ja nur die Teilsuche in den Klammern enthält.

Sie können den Speicher aber auch innerhalb des Suchstrings anwenden, dann allerdings mit der Syntax \1. Das beste Beispiel, das ich finden konnte, ist wohl die Suche nach Wortdoppelungen: \b(\w+)\s+\1\b Können Sie den Term schon entziffern? »Ein Wortanfang gefolgt von einem oder mehreren Buchstaben (die in \1 gespeichert werden), gefolgt von einem oder mehreren Leerzeichen, gefolgt von derselben Zeichenfolge wie vorne, gefolgt von einem Wortende«. Die Wortenden \b sorgen dafür, dass in »Der Fuß**ball ball**erte in der Gegend 'rum« nichts gefunden wird. Sonst hätten Sie ja auch in »Und dann nuckelte es sehr ruhig ganz zahm ahm Lolli« sieben Treffer.

Sehr praktisch ist \1 auch bei der Verarbeitung von HTML oder XML, da zusammengehörige Tags gefunden werden können: <([A-Z][A-Z0-9]*)[^>]*>.*?</\1> Sezieren wir den Suchstring:

- < Tags stehen immer in spitzen Klammern.
- ([A-Z][A-Z0-9]*) Ein Tag fängt immer mit einem Buchstaben an und kann dann eine beliebige Folge von Buchstaben und Ziffern enthalten. (Wir unterscheiden hier nicht zwischen Groß- und Kleinschreibung.) Die Klammern speichern den Tag in $1.
- [^>]* Falls der Tag Parameter hat, stehen diese noch vor dem »>«. Die Parameter ignorieren wir hiermit einfach.
- .*?< Zwischen dem anführenden und dem abführenden Tag stehen Null oder mehr Zeichen. Durch das ? ist dies eine faule Suche, die mit dem ersten »<« abbricht.
- /\1> Wir suchen ja das abführende Tag, daher muss direkt nach dem »<« ein »/« stehen, dann der gespeicherte Tag, gefolgt von »>«. Das /\ ist übrigens ein / und ein \, sprich: Wirklich ein ›\‹ und nicht das erste Zeichen von \d, der Wildcard für eine Ziffer.

Beispiel: Hochstellen

Es wird ein ASCII-Text geliefert mit #$tief#ein Wort$#, wobei ›ein Wort‹ automatisch tief gestellt werden soll:

#\$tief#([^$]+)\$#

ersetzen durch

$1

Erklärung:

#\$tief# findet einfach #$tief#, wobei das $ mit einem \ versehen werden muss.

([^$]+) findet das tiefzustellende Wort, in der Hoffnung, dass $ nicht in dem Wort vorkommen kann.

Falls doch ein $ vorkommen kann, müsste (.+?)\$# funktionieren. Das ist allerdings etwas langsamer als die obige Lösung.

Eine Warnung noch zur Kombination von Klammern und Wiederholungen: ([abch])+ und ([abch]+) finden beide bach, aber im ersten Fall steht in $1 lediglich »h« (jeder gefundene Buchstabe überschreibt den Inhalt von $1), im zweiten Fall korrekt »bach«.

Teilergebnisse zu speichern kostet Zeit und Platz. Bei einfachen Suchen-Vorgängen spielt das sicher keine Rolle. Wenn Sie aber richtig viel zu Suchen haben, setzen Sie besser die Klammern ein, die nicht in $1 speichern (?:[abch]+)

Nach Unicode-Zeichen suchen

Denken Sie an den Unterschied zwischen Unicode und Glyphen. Eine Ligatur ist zwar ein Zeichen, aber mehrere Buchstaben. Im Zweifelsfall schlagen Sie in der Glyphen-Palette den/die Unicodes nach und geben Sie im Suchfeld als \x{006F} ein. <006F> geht auch, wandelt aber gleich in die gesuchte Glyphe um.

Standardverhalten modifizieren

Es gibt drei Grundregeln, die bei normalen Suchen gelten:

- Groß-/Kleinschreibung wird unterschieden. d\.h\. findet also nicht ein D.h., das am Anfang eines Satzes steht. [dD]\.h\. wäre die Lösung mit unseren bisherigen Mitteln.
- . findet keine Absatzschaltung. .+ sucht also höchstens bis zum Ende des Absatzes.
- ^ findet den Anfang vom Absatz, $ das Ende.

Alle drei Regeln können mit (?ism) ein- und ausgeschaltet werden. (i: *insensitive*; s: *single-line-mode*; m: *multi-line-mode*)

- (?i)grep(?-i) ist toll findet grep ist toll und GREP ist toll, aber nicht grep ist Toll. Nach dem (?-i) unterscheidet die Suche wieder Groß-/Kleinschreibung.
- (?s).+Betrifft:(?-s).+$ Durch (?s) findet .+ auch die Absatzschaltungen. Dadurch wird der gesamte Text bis zum letzten Vorkommnis (. ist gierig) von »Betrifft:« markiert. Ohne das (?-s) würde das .+$ bis zum Ende des Textes gehen. So aber wird nur der Rest des Absatzes gefunden, in dem »Betrifft:« steht. *single-line-mode* betrachtet also den gesamten Text als eine Zeile/Absatz.
- ^. findet normalerweise den ersten Buchstaben des nächsten Absatzes. (?-m)^. findet den ersten Buchstaben des gesamten Texts. Den *multi-line-mode* auszuschalten, ist also quasi das Gleiche, wie den *single-line-mode* einzuschalten, nur dass (?-m) nur Auswirkungen auf ^ und $ hat und (?s) nur auf .
- Ein weiterer modifier ist *ignore whitespace* (?x). (?x) P D F findet auch PDF; die Leerzeichen werden halt einfach ignoriert. Dadurch lassen sich Suchterme wesentlich übersichtlicher

In Kürze

(?i)	Groß-/Kleinschreibung egal
(?-i)	Groß-/Kleinschreibung nicht egal
(?s)	. findet auch Absatzschaltungen.
(?-s)	. findet keine Absatzschaltungen.
(?m)	^ und $ berücksichtigen Absätze.
(?-m)	^ und $ berücksichten den ganzen Textfluss.
(?x)	Leerzeichen im Suchbegriff sind egal.
(?-x)	Leerzeichen sind nicht egal.

Beispiel: Nicht nur ein Wort in letzter Zeile

Per GREP können Sie das letzte Leerzeichen im Absatz durch ein geschütztes Leerzeichen ersetzen. Das führt dazu, dass noch mindestens eine Silbe des vorletzten Wortes mit in die letzte Zeile rutscht:

\s(?=[\w[:punct:]]+$)	ersetze durch ~S
\s	das Leerzeichen
(?=	positive Lookahead (siehe unten)
[\w[:punct:]]+	Wortzeichen oder ein beliebiges Satzzeichen wie ,.-!?
$	Absatzende
~S	Geschützter Leerraum

Danke an Peter Kahrel für diesen GREP.

schreiben. Aus <([A-Z][A-Z0-9]*)[^>]*>.*?</\1> kann dann (?x) < ([A-Z] [A-Z 0-9]*) [^>]* > .*? </ \1> werden.

In Kürze

a(?=b)	Finde a, wenn ein b folgt.
a(?!b)	Finde a, wenn kein b folgt.
(?<=a)b	Finde b, wenn direkt davor a steht.
(?<!a)b	Finde b, wenn direkt davor kein a steht.

Vorausschauend suchen

Kürzlich gab es bei HilfDirSelbst.ch die Anfrage, ob man durch Suchen/Ersetzen nicht alle Vorkommnisse von >> spationieren könnte, so dass die beiden > dichter beieinander stehen. Wenn man nun nach >> sucht und die Laufweite ändert, hat man ja auch die Laufweite des zweiten > geändert. Mit den Bordmitteln von CS2 geht diese Suche nicht, mit GREP durchaus.

GREP stellt Ihnen vorausschauendes und rückschauendes Suchen zur Verfügung, sowohl positiv als auch negativ.

(?= findet, wenn dem Ausdruck ein anderer Ausdruck folgt. >(?=>) findet also nur ein >, wenn es von einem > gefolgt wird.

(?<= findet, wenn der Ausdruck einem anderen folgt. (?<=>)> findet das zweite > von >>.

(?! findet, wenn dem Ausdruck ein bestimmter anderer Ausdruck **nicht** folgt. q(?!u) findet also jedes »q«, wenn danach kein »u« kommt. Mir fallen als Beispiel mathematische Formeln ein, da gibt es häufig »q«s ohne »u«.

(?<! findet, wenn dem Ausdruck ein bestimmter anderer Ausdruck **nicht** vorausgeht. Wenn es nach dem Adobe-Marketing ginge, dürften wir nicht von »InDesign« reden, sondern immer nur von »Adobe InDesign«. (?<!Adobe)InDesign findet alle InDesign, denen Adobe nicht voransteht.

Leerzeichen ignorieren

Dies ist ein schönes Beispiel für (?x).
Statt
(?<=\()([^)]+)(?=\))
können Sie auch
(?x) (?<=\() ([^)]+) (?=\))
schreiben.

Eine andere Anwendung wäre, alle Begriffe in Klammern auf kursiv zu stellen, also aus (kursiv) mach (*kursiv*), aber ohne die Klammern selbst schräg zu stellen. Formatierungsänderungen beim Suchen/Ersetzen wirken immer auf den gesamten gefundenen Text. \([^\)]+\) findet (kursiv), aber die Klammern sind im Gefundenen enthalten. (?<=\()([^)]+)(?=\)) hingegen findet (kursiv). Die Klammern müssen da sein, gehören aber nicht zum Gefundenen und werden nicht umformatiert.

Geht es leichter mit Leerzeichen?
(?x) (?i) \b [A-Z0-9._%+-]+ @ [A-Z0-9.-]+ \. [A-Z]{2,4} \b(?-i) (?-x)

Ein komplexes Beispiel zum Schluss

(?i)\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b(?-i)

Lesen Sie sich den Term durch. Sehen Sie schon, was hier gesucht wird? Gehen wir den Term durch:

(?i)	Groß-/Kleinschreibung egal. Das macht [A-Z] äquivalent zu [A-Za-z]
\b	Es wird ein ganzes Wort gesucht.
[A-Z0-9._%+-]+	Eine Reihe von Buchstaben, Ziffern oder einer Auswahl an Sonderzeichen wird gesucht.
@	Ein at-Zeichen

[A-Z0-9.-]+	Eine Reihe von Buchstaben, Ziffern, Punkt oder Strich
\.	Ein Punkt
[A-Z]{2,4}	Zwei bis vier Buchstaben
\b	Das Wort ist zu Ende.

Richtig, das stellt eine E-Mail-Adresse dar. Kann man das besser machen?

Als erstes fällt auf, dass keinerlei Plausibilität geprüft wird. asdf@asdf.ed wird auch gefunden. Wenn nur zugelassene Top-Level-Domains gefunden werden sollen, wird es viel komplizierter. Die Domain wird mit [A-Z0-9.+]+ gesucht. Dadurch, dass der Punkt mit in den Klammern steckt, findet er korrekt a@partners.adobe.com, aber auch a@adobe....de.

Stattdessen könnte man nach (?:[A-Z0-9-]+\.)+ suchen.

Peter Kahrel sagte mal, dass er bei jedem neuen GREP das Gefühl hat, er lerne das Thema gerade neu. Lassen Sie sich davon aber nicht abschecken. Notfalls helfen die freundlichen Leute der freundlichen Foren bestimmt gern weiter :)

Beispiel: Aus „Farbe-Palette" mach „Palette Farbe"
(\b[^-\s]+)-Palette
ersetzt durch
Palette $1

Erklärung:
[^...] findet alle Zeichen, die nicht ... sind.
-\s findet ein - oder einen beliebigen Leerraum.
+ findet eins oder mehrere.
( ) speichere das Gefundene in $1.

Es wird Ihnen häufig passieren, dass im durchsuchten Text Dinge stehen, mit denen Sie nicht gerechnet haben, zum Beispiel eine Mehrzahl: »Die Formate-Paletten«.

Dann müssen Sie den Suchbegriff anpassen, in diesem Fall: (\b[^-\s]+)-Paletten?

Sonderzeichen	Text	GREP
Tabulator	^t	\t
Absatzschaltung	^p	\r
Erzw. Zeilenschaltung	^n	\n
Seitenzahl	^#	~#
Aktuelle Seitenzahl	^N	~N
Nächste Seitenzahl	^X	~X
Vorige Seitenzahl	^V	~V
* Alle Variablen	^v	~v
Abschnittsmarke	^x	~x
* Marke: Verankertes Objekt	^a	~a
* Marke: Fußnote	^F	~F
* Marke: Index	^I	~I
Punkt •	^8	~8
Caret ^	^^	\^
Schrägstrich \	\	\\
Copyright-Symbol ©	^2	~2
Ellipse …	^e	~e
Tilde ~	~	\~
Paragraph-Symbol §	^7	~7
Registered-Trademark-Symbol ®	^r	~r
Abschnitts-Symbol ¶	^6	~6
Trademark Symbol ™	^d	~d
Geviertstrich	^_	~_
Halbgeviertstrich	^=	~=
Bedingte Trennung	^-	~-
Geschützter Trennstrich	^~	~~
Beliebiger Unicode	<0041>	<0041>
Beliebiger Unicode		\x{0041}
Leerzeichen	**Text**	**GREP**
Geviert	^m	~m
Halbgeviert	^>	~>
Drittelgeviert	^3	~3
Viertelgeviert	^4	~4
Sechstelgeviert	^%	~%
Ausgleichszeichen	^f	~f
24tel Geviert	^|	~|
Geschütztes Leerzeichen	^S	~S
Geschütztes Leerzeichen, feste Breite	^s	~s

Sonderzeichen	Text	GREP
Achtelgeviert	^<	~<
Ziffernleerzeichen	^/	~/
Interpunktionsleerzeichen	^.	~.
Zwischenablage	**Text**	**GREP**
Zwischenablage, formatiert	^c	~c
Zwischenablage, unformatiert	^C	~C
Umbruchzeichen	**Text**	**GREP**
Absatzumbruch (= ^p ?)	^b	~b
Spaltenumbruch	^M	~M
Rahmenumbruch	^R	~R
Seitenumbruch	^P	~P
Umbruch ungerade Seiten	^L	~L
Umbruch gerade Seiten	^E	~E
Bedingter Zeilenumbruch (Zero-Width-Space)	^j	~a
Tab rechter Einzug	^y	~y
Einzug bis hier	^i	~i
Verschachteltes Format beenden	^h	~h
Verbindung unterdrücken (Nonjoiner)	^k	~k
Variablen	**Text**	**GREP**
Laufende Kopfzeile (Absatz)	^Y	~Y
Laufende Kopfzeile (Zeichen)	^Z	~Z
Benutzerdefinierter Text	^u	~u
Letzte Seitenzahl	^T	~T
Kapitelnummer	^H	~H
Erstellungsdatum	^S	~S
Änderungsdatum	^o	~o
Ausgabedatum	^D	~D
Dateiname	^l (klein L)	~l (klein L)
Wildcards	**Text**	**GREP**
*Ziffer	^9	\d
*Alles außer Ziffern		\D
* Buchstabe	^$	[\l\u]
* Zeichen (Beim GREP-Ersetzen wird ein . eingefügt)	^?	.
* Weißraum (Leerzeichen oder Tab)	^w	\s

Sonderzeichen	Text	GREP
* Kein Weißraum		\S
* Wortzeichen (also Zeichen, die in Wörtern vorkommen)		\w
* Kein Wortzeichen		\W
* Großbuchstabe		\u
* Kein Großbuchstabe		\U
* Kleinbuchstabe		\l
* Kein Kleinbuchstabe		\L
Der gefundene Text		$0
Gefundener Text 1..9		$1..$9
* Kanji	^K	\K
Positionen	**Text**	**GREP**
Wortanfang		\<
Wortende		\>
Wortgrenze		\b
Keine Wortgrenze		\B
Absatzanfang		^
Absatzende		$
Anfang Textfluss		\A
Ende Textfluss vor Return		\Z
Ende Textfluss nach Return		\z
Wiederholungen	**Text**	**GREP**
Ein- oder keinmal		?
Kein- oder mehrmal		*
Ein- oder mehrmal		+
Kein- oder einmal (kürzeres Ergebnis)		??
Kein- oder mehrmal (kürzestes Ergebnis)		*?
Ein- oder mehrmal (kürzestes Ergebnis)		+?
Genau n Mal		{n}
Mehr als n Mal		{n,}
Zwischen n und m Mal		{n, m}
Strukturelles	**Text**	**GREP**
Teilbegriffe suchen mit Speicher		( )
Gespeicherten Teilbegriff ersetzen		$1..$9
Gespeicherten Teilbegriff noch mal finden		\1..\9
Teilbegriffe suchen ohne Speicher		(?:)
Eines der Zeichen in der Klammer		[…]
Ein Zeichen, das nicht in der Klammer steht		[^…]
Logisches Oder		|
Umgebung	**Text**	**GREP**
Positive Lookbehind		(?<=)
Negative Lookbehind		(?<!)
Positive Lookahead		(?=)
Negative Lookahead		(?!)
Suchparameter	**Text**	**GREP**
Groß-/Kleinschr. ignorieren		(?i)
Groß-/Kleinschr. beachten		(?-i)
Multiline an		(?m)
Multiline aus		(?-m)
Single-line an		(?s)
Single-line aus		(?-s)
Ignore Whitespace an		(?x)
Ignore Whitespace aus		(?-x)
Sonderklassen	**Text**	**GREP**
* Alphanumerisches Zeichen [a-zA-Z0-9]		[[:alnum:]]
* Zeichen aus dem Alphabet [a-zA-Z]		[[:alpha:]]
* Weißraum (Leerzeichen oder Tab) [\t]		[[:blank:]]
* Kontrollzeichen		[[:control:]]
* Sichtbare Zeichen [^ \t\n\r]		[[:graph:]]
* Sichtbare Zeichen und Leerzeichen [^\t\n\r]		[[:print:]]
* Satzzeichen [.,!?:;…]		[[:punct:]]
* Zeichen mit einem Code größer 255		[[:unicode:]]
* Hexadezimale Ziffer [0-9a-fA-F]		[[:xdigit:]]
Mit * gekennzeichnete Begriffe können nur beim Suchen verwendet werden.		

▲ **Tabelle 3.1**
Alle Metazeichen für die Suche