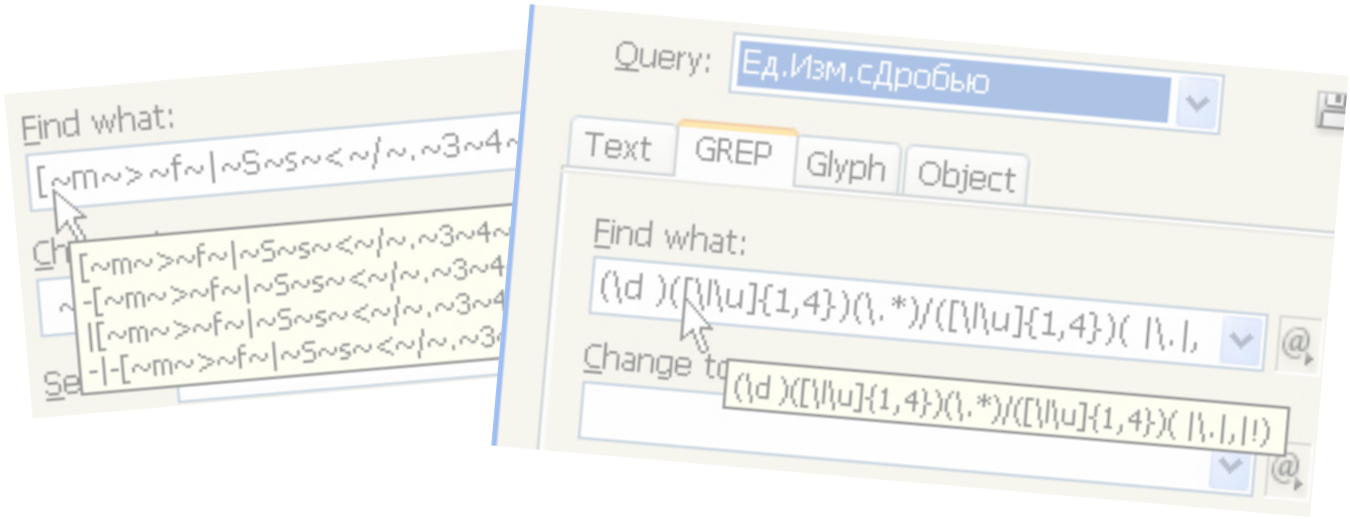
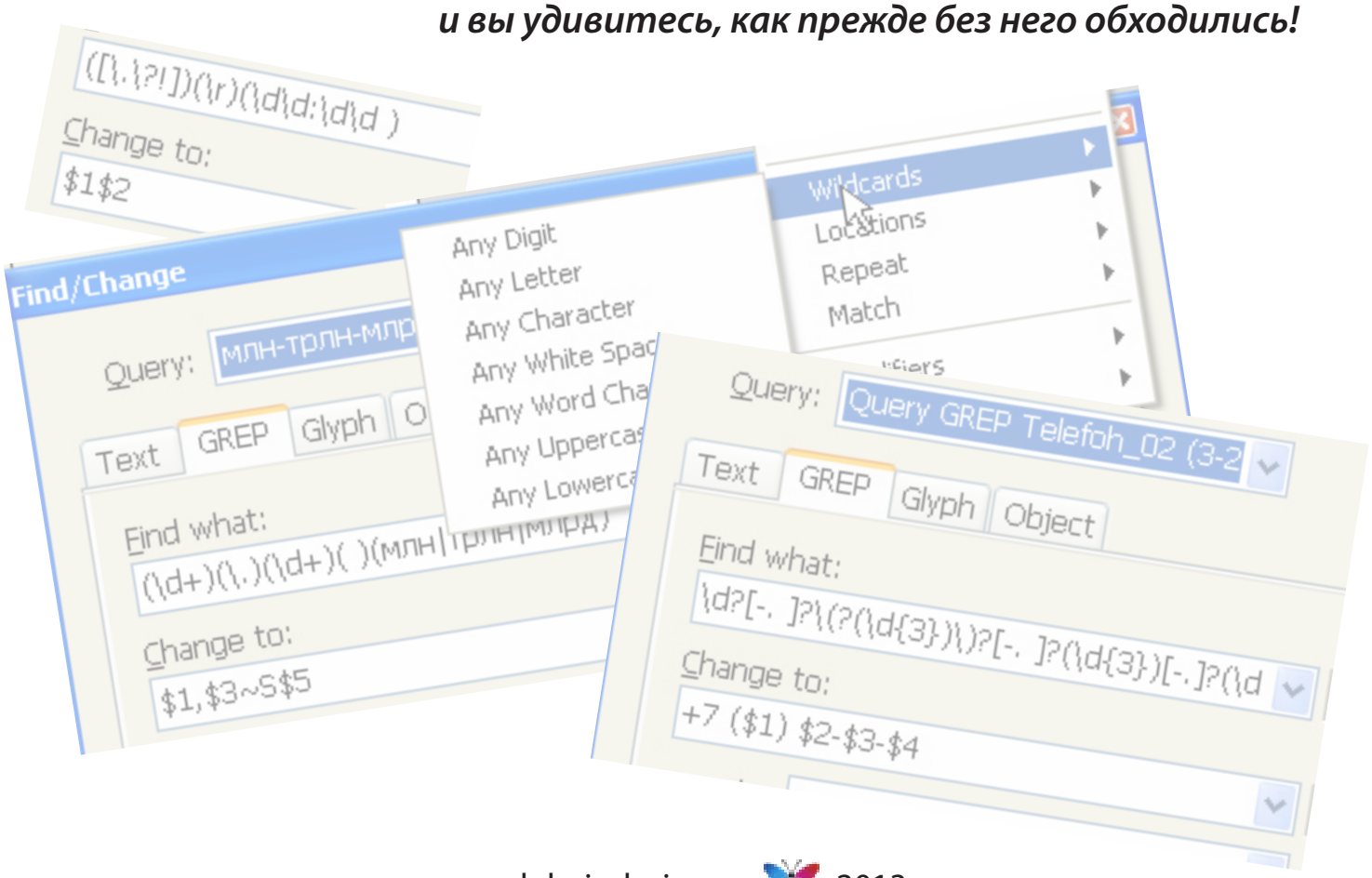


GREPоводство

Регулярные выражения в InDesign

Потратьте несколько часов на изучение GREP — и вы удивитесь, как прежде без него обходились!



adobeindesign.ru 2013

На мой взгляд, в русскоязычной литературе мало публикаций, доступно объясняющих работу такого замечательного инструмента в составе InDesign, как GREP. Есть множество сухих таблиц, где собраны только коды операций и их названия, есть специализированные форумы, где встречаются россыпи страниц с «вопросами и ответами»… Но только на английском языке, к сожалению, я находил толковые публикации, после прочтения которых становятся ясны и те таблицы, и ответы на форумах, и где — что немаловажно! — внятным образом изложены задачи, для решения которых верстальщику понадобился GREP.

Именно из-за очевидного дефицита информации на русском языке мне захотелось сделать собственную книгу о GREPе. Я говорю «книга», но по сути это будет просто рассказ о возможностях этого инструмента. И сделать её так, чтобы ведомый тобой читатель не утопал в деталях в надежде сперва всё хорошенько изучить, а потом начать применять — такой подход ещё никогда не приводил к успеху. Нет, дать небольшой экскурс в описание инструмента, а потом сразу пробовать применять инструмент, делать эту практику своим опытом, тогда последующие объяснения наложатся на то, что уже знаешь.

Так что эта маленькая книга не претендует на то чтобы считаться библией грепа. Но она точно поможет сделать GREP родным, понятным инструментом, даже если до сих пор вы совершенно не разбирались в метасимволах, а слова вроде POSIX или Positive Lookbehind звучали не яснее любой другой абракадабры.

Идеей, достойной повтроения, было решение, встреченное в книге Майкла Марфи «GREP in InDesign CS3» — подробно объяснить словами, что будет делаться, и первый сложный пример раскрасить цветами, чтобы начинающий понял все детали обработки. Такой подход к началу объяснения этого инструмента повторён и тут, но вместе с тем эта книга содержит информации намного больше, нежели та книга Марфи. Уверен, что после усвоения приведённых тут примеров вы ощутите настоящее удовольствие от открывшихся возможностей и будете пополнять свою коллекцию приёмов использования GREP. Конечно, вместе с руководством вы получите и полные перечни всех кодов, и ряд ссылок на полезные сайты… Но главная моя задача состоит в том, чтобы собирание информации с прицелом на неопределённое «когда-нибудь» уступило место собственному опыту в применении этого инструмента.

Лучше уметь применять хотя бы небольшую долю возможностей GREP, нежели забить целый шкаф распечатками, не помня из них ни одной строчки.

Копите умения, а не бумагу!

Михаил Иванюшин
m.ivanyushin@gmail.com
Вопросы, идеи — пишите.

Я благодарен первым читателям набросков «GREPоводства» —
Татьяне Прокофьевой, Борису Кащееву и Алексею Чмелю
за их помощь, благодаря которой черновик превратился в книгу.

GREP в ИнДизайне

Вы понимаете, что такое GREP? Если для вас это пока больше мистика, непостижимая головоломка реализации экзотических процедур, совершенно вам непонятных, то эта книга — грамматика разработки эффективных преобразований.

Такая расшифровка аббревиатуры, конечно, шутка, но, как во всякой шутке, здесь есть и доля правды.

Что такое GREP?

Майкл Марфи (Mikhael Murphy) считает, что GREP — это сокращение английской фразы «general regular expression parser» — «анализатор общих регулярных выражений».

А вот Питер Карел (Peter Kahrel) в своей книге «GREP in InDesign CS3» приводит другое толкование этой аббревиатуры: «general regular expression print» — «вывод общих регулярных выражений».

Такое объяснение перекликается с определением из Википедии, что это выборочное сокращение фразы «search globally for lines matching the regular expression and print them» — «искать строки, соответствующие регулярному выражению и выводить их».

И мы будем недалеки от истины, если скажем, что GREP это генерация различных эффективных преобразований.

Что же такое *регулярное выражение*?

Слово «регулярный» в данном контексте не следует понимать как «происходящий с определёнными интервалами», а надо вспомнить, что оно происходит от латинского regula, «правило». Таким образом, «регулярное выражение» означает «выражение, содержащее правила или состоящее из правил».

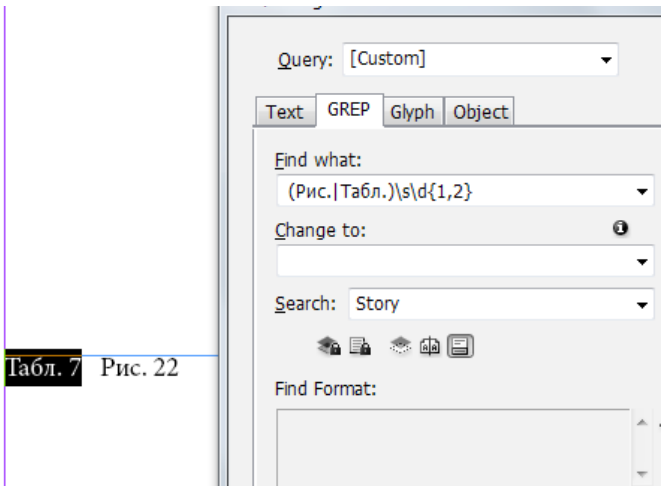
В контексте GREP, «регулярное выражение» — это описание шаблона или условий, которым должен отвечать искомый текст. И, в отличие от обычного текстового поиска, такое условие может вовсе не содержать конкретных букв, или слов, или символов обрабатываемого текста.

Можно искать строчные, прописные, гласные или согласные буквы, цифры, знаки пунктуации и пр. Мощь этого инструмента условных обозначений в том, что он позволяет выполнять поиск/замену не конкретного слова или буквы, а разного текста, соответствующего заданным критериям.

Пусть вас не смущает, что инструкции регулярных выражений выглядят, как криптографическая запись: такая нотация осталась неизменной со времён первых языков программирования. Как только вы оцените возможности этого «языка условий», странные и запутанные на первый взгляд инструкции обретут смысл и перестанут вас пугать.

Поиск поиску рознь

При обычном поиске (вкладка Text) ищется точное совпадение. Например, ищется «молоко» и заменяется на «масло». И при этом поиске нельзя настроить запрос так: найти «масло» *или* «молоко» и заменить на «продукт», и это в процессе одной операции поиска/замены. А для GREP такой поиск — обычное дело.



Условие «*или*» в GREP обозначается вертикальной чертой. Это и есть «условный» поиск, т.е. поиск через условие.

А можно ли с помощью GREP найти, скажем… целый абзац? Чтобы определённо ответить на этот вопрос, надо проверить, можно ли описать обобщённый абзац — т. е. любой абзац! — в виде чёткого правила, в виде шаблона. Например, так:

«Абзац — это последовательность символов, которая начинается после предыдущего символа перевода строки (или стоит в абсолютном начале текстового фрейма) и заканчивается также символом перевода строки».

Очень скоро вы увидите, как это выражение перевести в понятный для машины вид. Сейчас же важно обратить внимание вот на что: в определении абзаца не фигурируют ни конкретный текст, ни объём текста, а только определена его структура: «*начало абзаца… далее идёт какой угодно текст… пока не встретится следующий символ конца абзаца*». С помощью подобных шаблонов GREP и осуществляет поиск, для него имеет значение структура искомого, где количество — лишь один из возможных параметров.

Шаблоны

Шаблоны есть везде, знаете вы об этом или нет. И вы будете их видеть и мыслить ими, как только у вас начнет получаться работать с этим инструментом. Даже при обычном поиске текста можно частично использовать идею шаблона. Например, вы можете запретить учитывать регистр букв и запретить искать только слово целиком. Тогда запрос «кофе» в поисковой строке найдёт и «Кофе», и «КОФЕ», и «кофемолка». Опция «поиска целого слова» исключит слова, перед которыми и после которых стоят пробел, символ перевода строки или знак пунктуации. И слово «кофемолка» найдено не будет, потому что не соответствует этому простому шаблону. Во вкладке Text количество шаблонов минимально и не может быть пополнено.

А в GREP вы можете использовать и такие же шаблоны, как в Text (только оформленные немного иначе), и определять свои собственные шаблоны и/или условия. Более того, шаблоны можно использовать и в строке «Заменить», что позволяет обрабатывать найденный текст — удалять, переставлять местами, преобразовывать. То есть редактировать.

Как и из чего строятся шаблоны? Они создаются из специальных символов (или сочетаний символов), называемых *метасимволами*.

Метасимволы

В большинстве случаев GREP-поиск выглядит, как обычный поиск текста. Обычно искомые буквы и те, что в шаблоне, совпадают. Т.е., если вы ищете букву «t», то и найдете букву «t», но если в поле поиска указать \t, то результатом поиска будут знаки табуляции. Обратная наклонная черта превращает букву «t» в метасимвол – комбинацию символов, представляющую собой новый знак с особым значением.

Метасимволы есть как в обычном поиске, так и в GREP-поиске. В обычном поиске метасимволы используются, в частности, для таких случаев, когда сам символ не может быть введён в окошко поиска или замены. Например, это знак табуляции, или знак перевода строки. Если просто нажать клавишу табуляции при открытом диалоговом окне, то сработает другое значение этой клавиши — перемещение курсора по полям диалога. Поэтому, если вам понадобилось найти именно символ табуляции в тексте, в окошко поиска надо ввести ^t.

Наверняка вы много раз видели эти метасимволы при работе с поиском/заменой на вкладке Text окна Find/Change. В GREP-поиске метасимволы более мощные.

Многие метасимволы состоят из двух символов: первый из них модификатор. При текстовом поиске модификатор — это символ ^. Так, табуляция обозначается как ^t, обычный перевод строки ^p, принудительный (Shift+Enter) — ^n, и т. п. При GREP-поиске в качестве модификаторов используются \ или ~. Символ \ — это стандартный модификатор для GREP выражений, а тильда ~ используется для обозначения уникальных символов именно в InDesign (шпации, маркер привязки объекта, нумератор страницы и пр.).

Однако некоторые метасимволы — это один символ. Например, точка . — это любой знак, кроме обоих вариантов перевода строки (обычного и принудительного). Поэтому, если надо указать, что ищется именно точка, то перед ней должен в окне поиска (*и только в окне поиска, но не в поле замены!*) быть знак обратной наклонной черты: \. Поначалу это непривычно, поскольку обычно обратная наклонная черта служит для обозначения метасимвола, а тут этот знак используется для превращения метасимвола в обычный символ.

GREP метасимвол	Назначение
. точка	Любой знак
^	Начало абзаца
\^	Начало группы исключения
$ знак доллара	Конец абзаца
* звёздочка	Повтор ноль или больше раз
\d	любая цифра
\D	любой символ, кроме цифр
\s	любая шпация
\S	любой символ, кроме шпаций
\b	граница слова
\B	граница слова из рассмотрения исключена
? знак вопроса	Повтор ноль или один раз
+ знак плюс	Повтор один и более раз
| вертикальная линия	Операция 'ИЛИ'
(открывающая круглая скобка	Начало поиска
) закрывающая круглая скобка	Конец поиска
[открывающая квадратная скобка	начало определения набора знаков
] закрывающая квадратная скобка	конец определения набора знаков

Таблица на этой странице идентифицирует одиночные метасимволы и объясняет их назначение.

Типы метасимволов

Разные метасимволы выполняют разные функции, и они могут быть упорядочены в разные наборы. Простейший из них — подстановочные символы: . — любой символ, \d — любая цифра, \s — любой пробел. В большинстве случаев из названия элемента набора ясно, что он делает, т.е. это заменитель места символов различных типов.

Другой класс метасимволов — это управление повторением числа совпадений. Используются такие метасимволы: ? /нет совпадения или одно совпадение/, * /нет совпадения или много совпадений/, + /один и много совпадений/. Сочетая их с метасимволами заменяющих символов, создаются универсальные запросы, например, .+ — найти любой знак, появляющийся один и более раз, \d* — найти любую цифру, появляющуюся ноль или более раз. (Именно цифру, не число, почувствуйте разницу)

Поначалу формулировка 'ноль или более раз' звучит непривычно — в обычной речи мы не скажем «найдено ноль раз», мы скажем «не найдено вообще». Но как только вы увидите результат выполнения запросов, вы привыкнете к этой фразе из жаргона программистов.

Повторением можно управлять более точно, указывая число совпадений. Это число или числа должны быть заключены в фигурные скобки. Например, \d{3} найдёт три любых подряд идущих цифры; \d{3,6} найдёт по крайней мере три цифры, но не больше шести.

Следующий класс метасимволов определяет *положение*, такое как начало слова \< , начало абзаца ^ , конец слова \> , конец абзаца $, \A – начало статьи (буква А должна быть прописной!), \z или \Z – конец статьи. И снова сочетание этих метасимволов с другими позволяет создавать уникальные и мощные запросы. Например, запрос ^\d+\. найдет все абзацы, начинающиеся с цифры или цифр, после которых стоит точка. (Точнее, этот запрос найдет именно эти цифры и точку в начале абзацев.)

Уникальность инструмента GREP и в том, что в запросах можно управлять совпадением символов. Диапазон искомых знаков заключается в квадратные скобки. Например, [аеёиоуыэюя] позволит искать гласные буквы в русском языке. Кроме того, и это часто используется, можно собрать в группу символы, которые должны пропускаться при поиске, так называемую группу исключения. Вот такая запись [^0-9] определяет, что цифры от 0 до 9 будут пропускаться при поиске. Такое исключение выполняется при помощи метасимвола ^, который действует на все символы, стоящие в квадратных скобках. Кроме этого существуют стандартные группы символов, они собраны в меню Posix (см. с. 8).

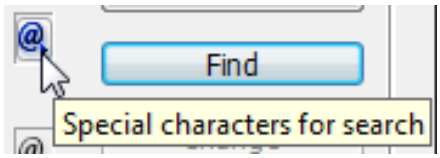
При помощи метасимвола | задаётся условие 'ИЛИ', например, GREP-поиск слов 'кофе', 'сок', 'чай' выглядит так: \<кофе\>|\<сок\>|\<чай\>. Перед каждым

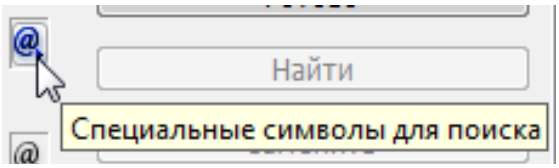
из этих слов стоит метасимвол указания границы слова, это исключает нахождение 'сок' в слове 'кусок', или 'чай' в слове 'скучайте'. Если ищутся какие-то уникальные слова, не являющиеся частью других слов, этим метасимволы \< и \> можно опустить.

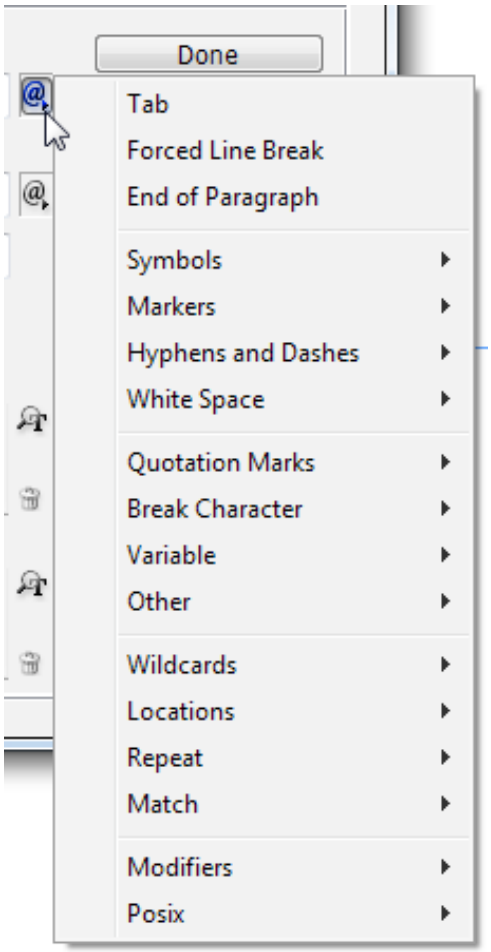
Другие заменяющие метасимволы определят и заменят подшаблоны (или подвыражения), которые могут быть использованы в запросе или операции поиска/замены. Например, для поиска номеров телефона, таких как 8 111 222-3333, можно использовать шаблон \d{1}\s\d{3}\s\d{3}-\d{4}. Но чтобы можно было работать с найденным текстом, надо, чтобы каждый подшаблон был в круглых скобках (\d{1})\s(\d{3})\s(\d{3})-(\d{4}). Такие подвыражения играют ключевую роль в поле Change (Заменить) в GREP-поиске. Каждый подшаблон имеет свою очередность, в соответствии с которой он стоит в окне поиска. В предыдущем примере подшаблон \d{1} адресуется как $1, что означает «найденный текст 1» (первый из трёх подшаблонов описания телефонного номера). Три первые цифры номера определяются как $2, последние четыре – как $4. Эти подшаблоны в окне «Заменить» записываются так: $1 $2 $3-$4.

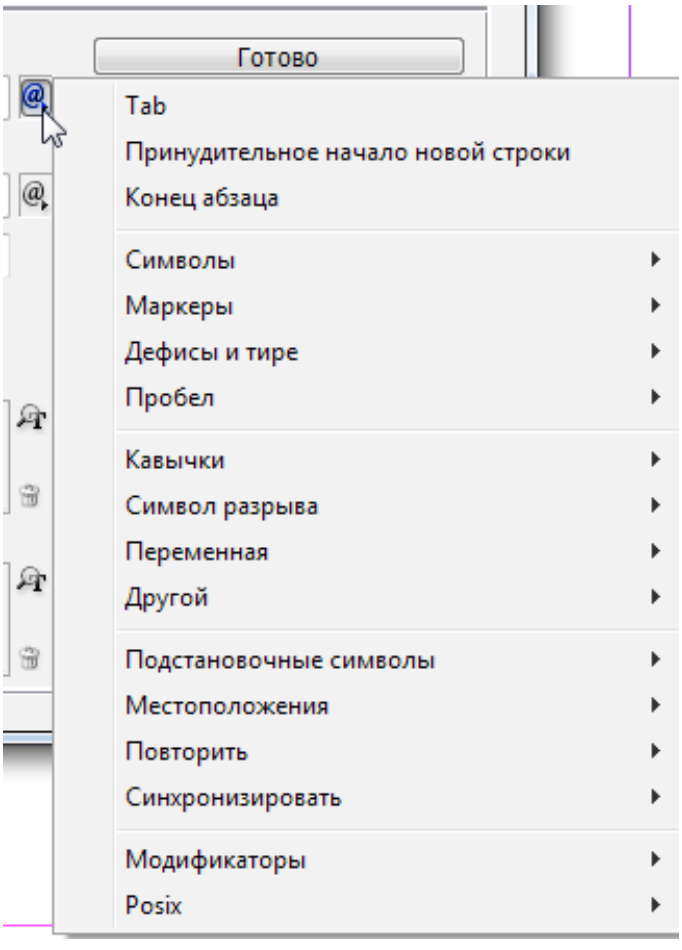
Специальные символы для поиска

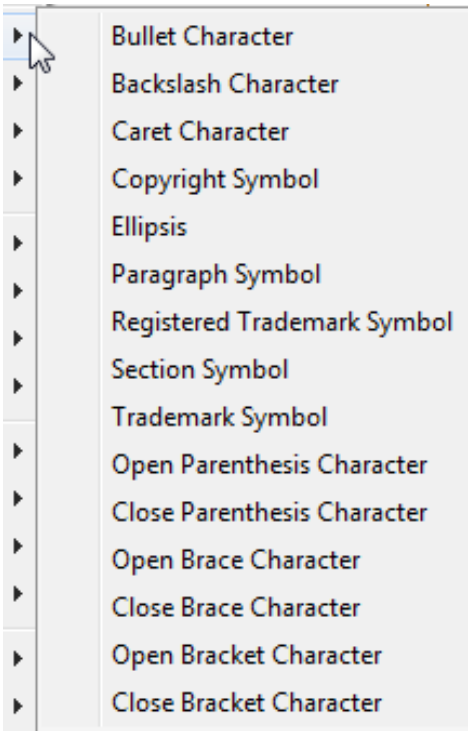
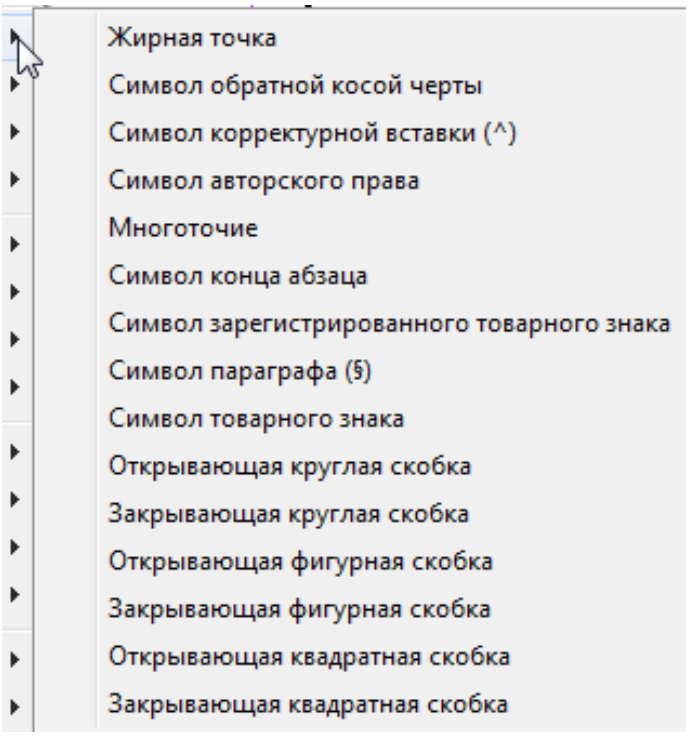
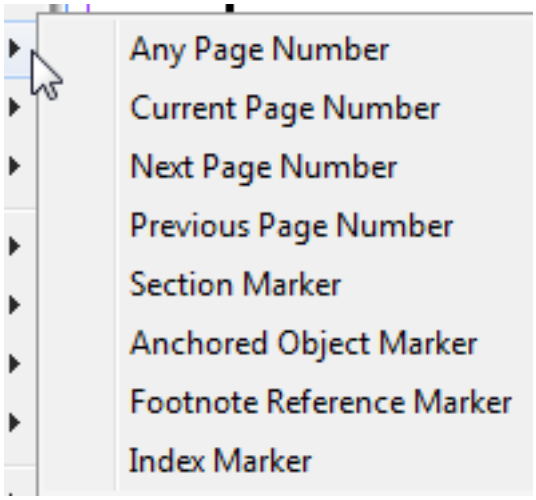
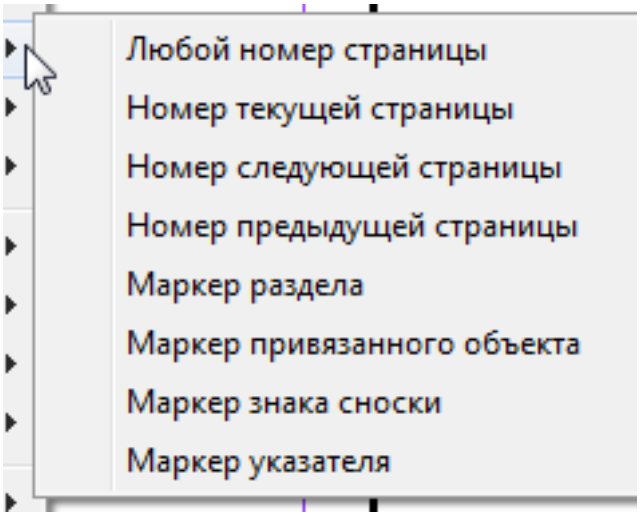
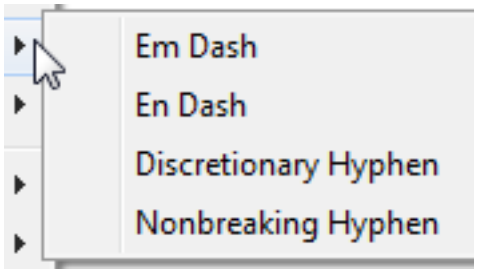
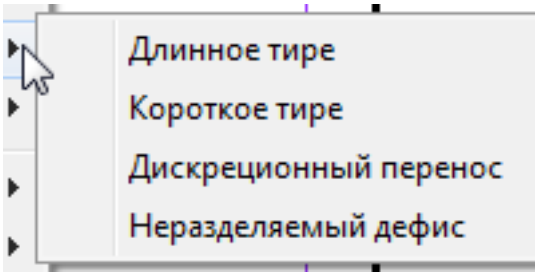
Вряд ли кому понравится идея запоминать все специальные знаки, поэтому в интерфейсе окна Поиска/замены есть выпадающее меню доступа ко всем этим знакам. Нажмите на @ и оно откроется. Все знаки упорядочены по группам. Специальные символы в этой части меню те же, что в текстовой части Поиска/замены. Однако нужно помнить, что модифицирующие символы, используемые в GREP-поиске, — это \ или ~, и никогда не ^. Ниже приведены все меню знаков для русской и английской версий программы.

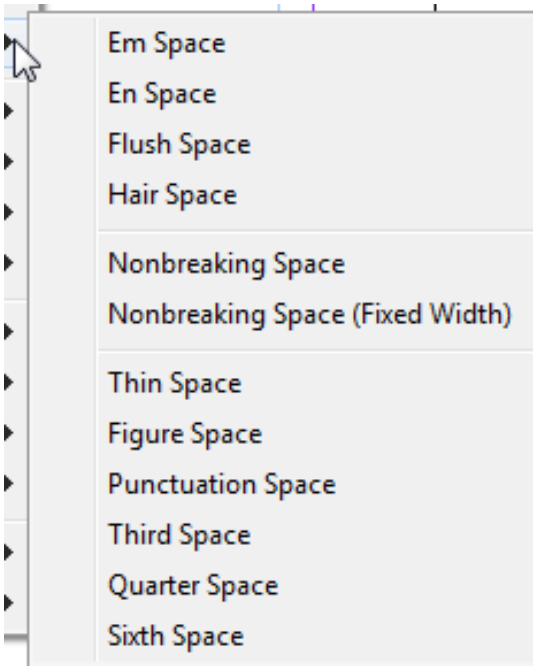
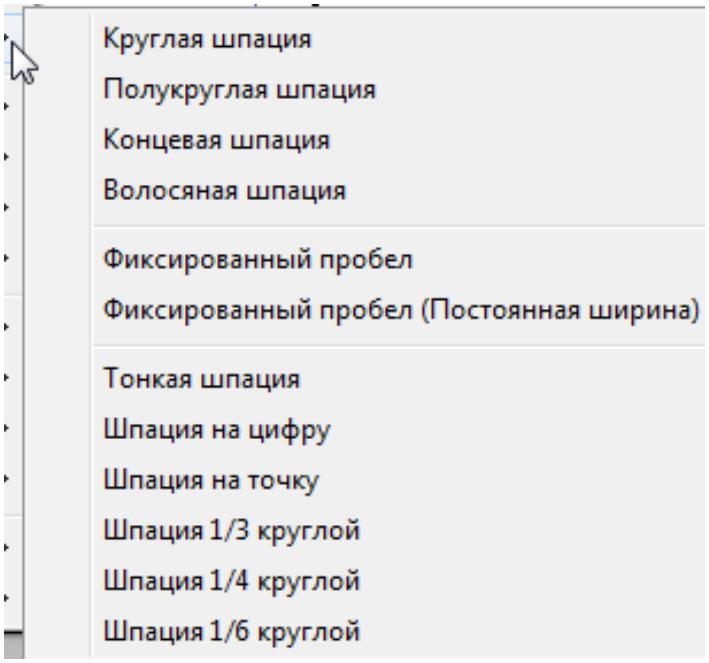
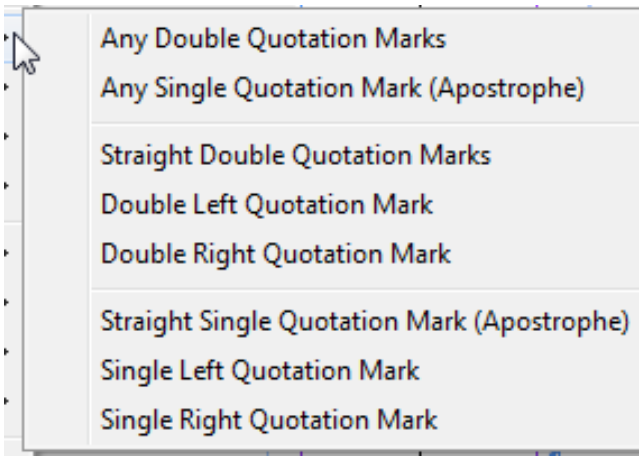
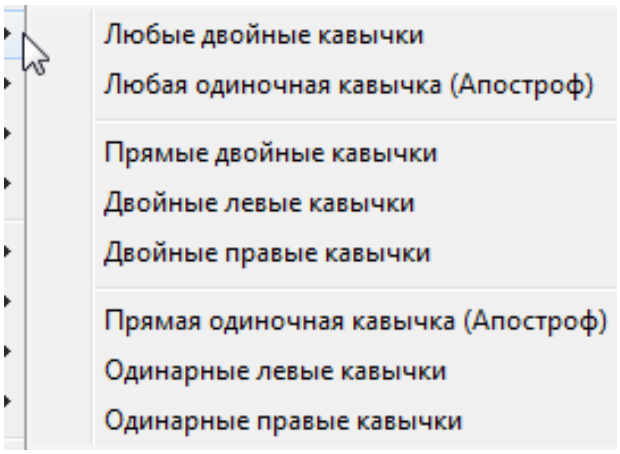
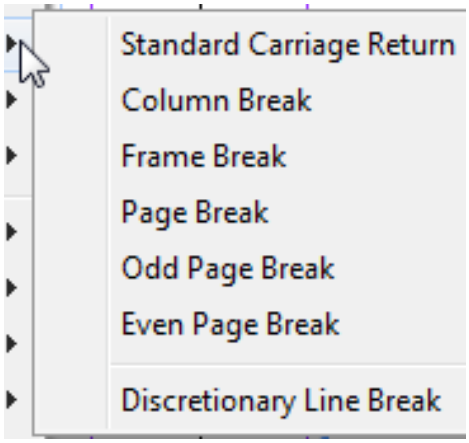
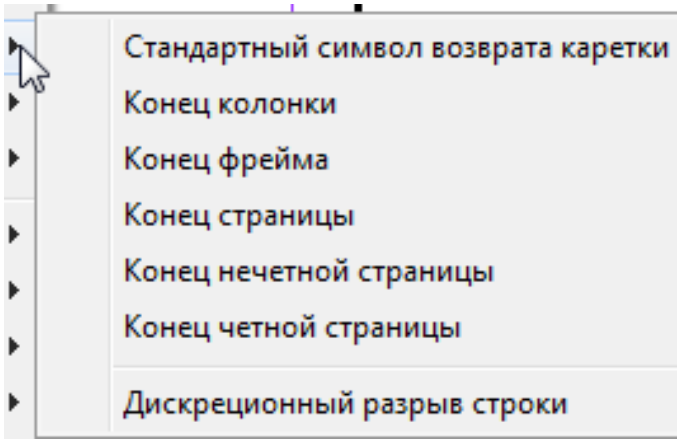


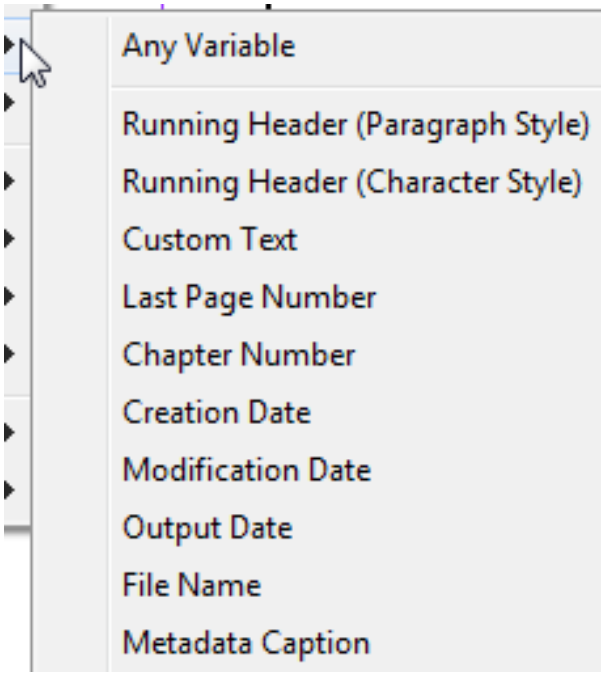
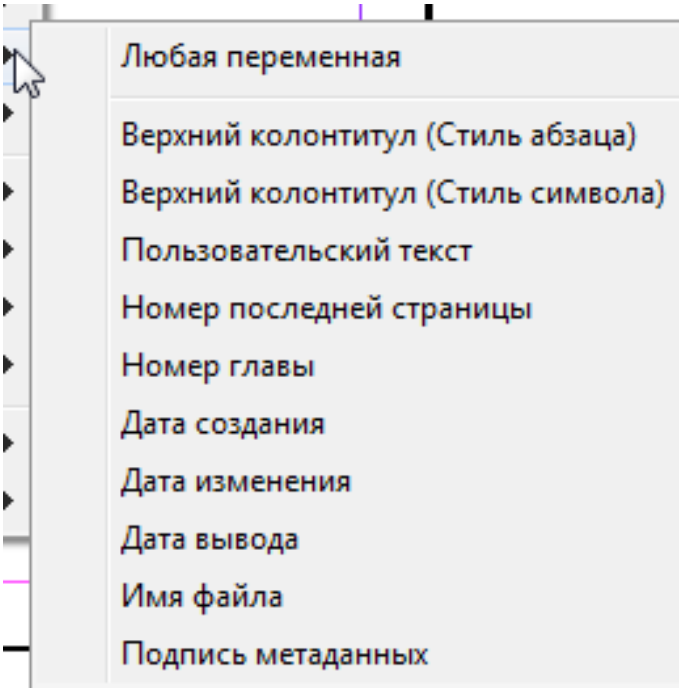
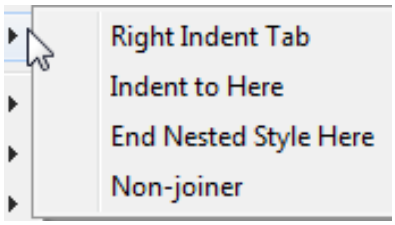
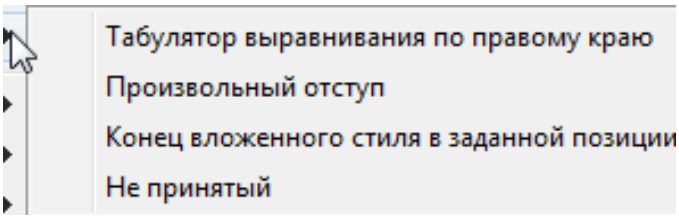
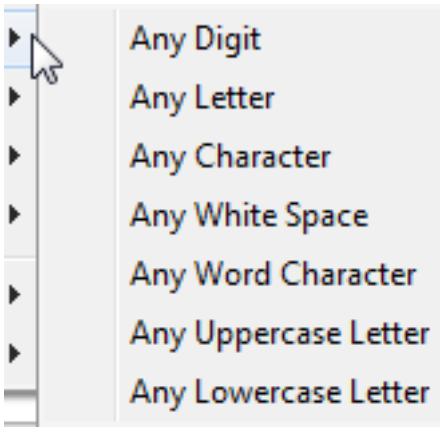
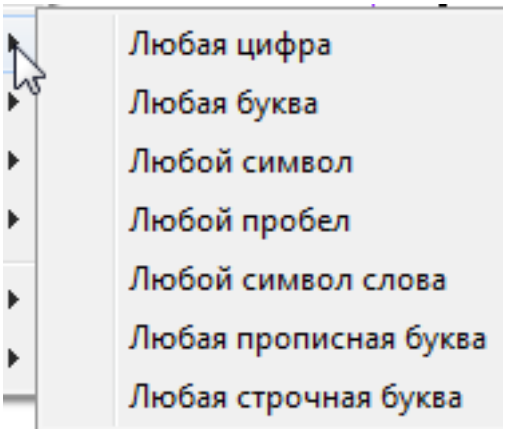
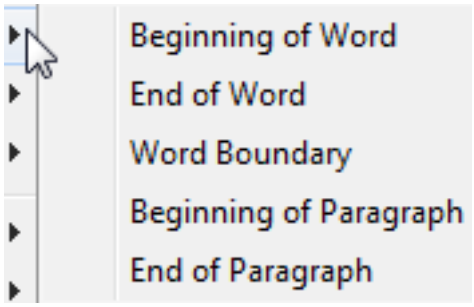
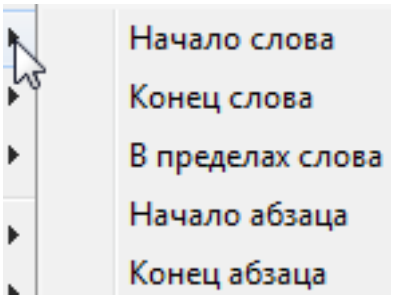


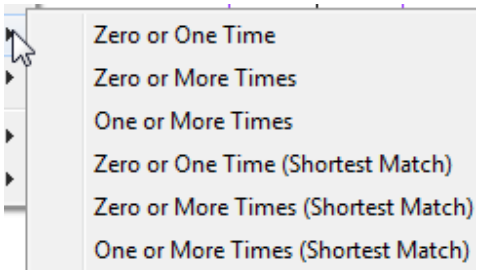
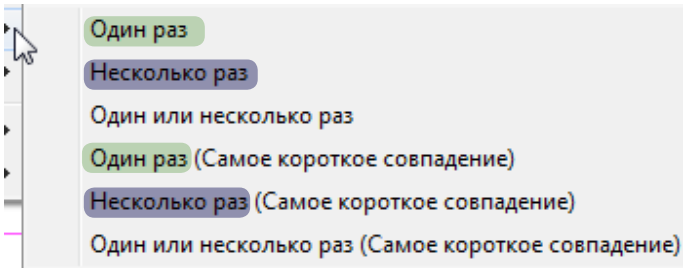




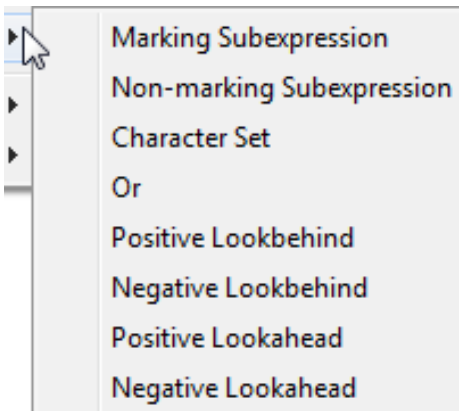
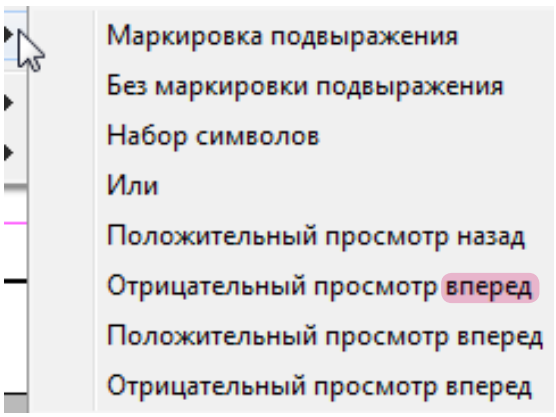
Symbols	Символы
	
Markers	Маркеры
	
Hyphens and Dashes	Дефисы и тире
	

White Space	Пробел
	
Quotation Marks	Кавычки
	
Break Character	Символ разрыва
	

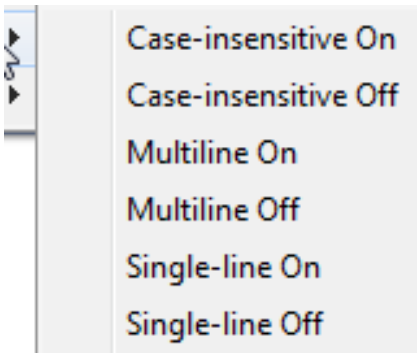
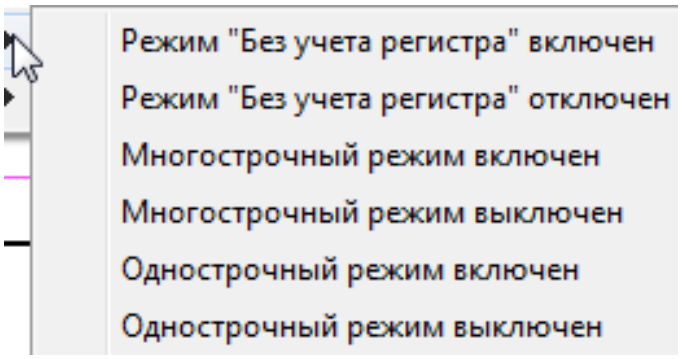
Variable	Переменная
	
Other	Любой
	
Wildcards	Подстановочные символы
	
Locations	Местоположения
	

Repeat	Повторить
	

● Ошибка локализации. Должно быть написано «Ни разу или один раз».
● Ошибка локализации. Должно быть написано «Ни разу или несколько раз».

Match	Синхронизировать
	

● Ошибка локализации. Должно быть написано «назад».

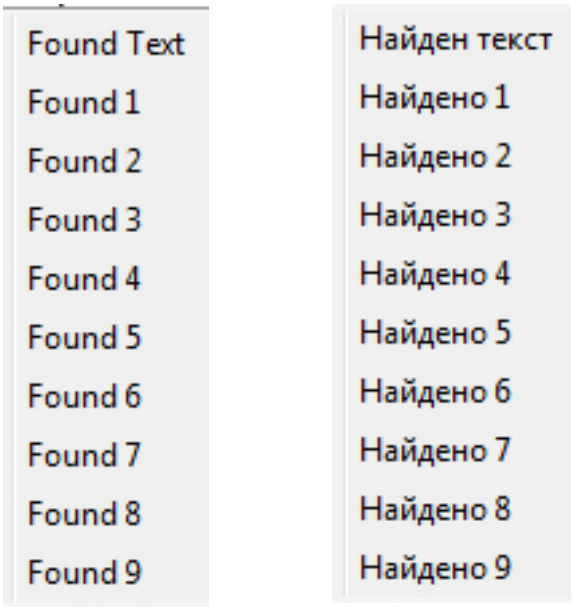
Modifiers	Модификаторы
	

Posix	**Стандартные группы знаков:**
[[:alnum:]]	алфавитно-цифровые символы
[[:alpha:]]	только буквы
[[:digit:]]	только цифры
[[:lower:]]	только буквы в нижнем регистре
[[:punct:]]	знаки пунтуации — точки, запятые, дефисы, тире, все скобки, кавычки и пр.
[[:space:]]	шпации, табуляция и символы перевода строки
[[:upper:]]	только буквы в верхнем регистре
[[:word:]]	только слова
[[:xdigit:]]	шестнадцатеричные числа
[[=a=]]	группы отдельных знаков: [[=a=]] для àáâãäå, [[=o=]] для òóôõö и т.п.

Специальные символы для замены

Поле 'Change to' (Заменить) также имеет выпадающее меню, но там меньше опций в сравнении с подробным меню, описанным выше. Многие из позиций (знак подстановки, позиции, совпадение, повтор и т.п.) тут недоступны. Например, нельзя заменить найденную цифру на любую. Действительно, такая задача смысла не имеет.

Поэтому опции Wildcard, Locations, Repeat, Match, Modifiers and Posix тут отсутствуют. В отличие от предыдущего меню есть опция 'Найден текст' / 'Found Text', там девять позиций от Found1 до Found9.



Found 0, обозначается как $0 — это весь текст, найденный по запросу, Found 1($1) — (Found9) $9 — подшаблоны текста. Если в поиске надо иметь больше 9 подшаблонов, то поиск надо выполнять с несколькими запросами.

Если вы по ошибке укажете $10, $11, запрос будет обработан, но в первом случае после $1 будет добавлен ноль, после второго $1 — единица. ***Не наступайте на эти грабли!***

Учёт регистра

В любом GREP-запросе можно как учитывать регистр искомого текста, так и отключать это анализ. Чтобы выключить учёт регистр, используется метасимвол '?i' , для включения — '?-I'. Введите в нашем самом первом примере в строке GREP-поиска '(?i)молоко|масло' , и вы найдёте эти слова, начинающиеся со строчной и прописной букв. Запрос '(?-i)молоко|МАСЛО' обусловит поиск только тех слов, которые абсолютно совпадают с заданными в шаблоне.

Определение диапазона поиска

GREP-поиск старается всегда найти как можно больше совпадений. Это часто называется «жадный поиск». Допустим, в абзаце ищется текст в круглых скобках. Если задаться целью создать запрос «найти открывающую скобку, затем несколько символов и закрывающую скобку», то такой запрос мог бы выглядеть так: \(.+\) . Обратите внимание, что перед скобками стоит метасимвол \ , чтобы эти скобки рассматривались как скобки, а не GREP-метасимволы.

Запись .+ предписывает искать один или больше любых символов.

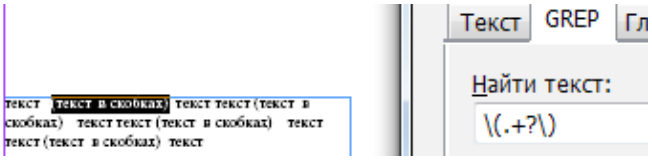
И вот как этот запрос будет работать. Если в абзаце несколько раз встречается текст в круглых скобках, то будет выбрана первая открывающая скобка, весь текст за ней, включая последнюю закрывающую скобку. Это и объясняет название этого запроса — «жадный».

Чтобы умерить аппетит, и искать текст в рамках одной пары скобок, надо пользоваться другим запросом: \(.+?\) .

Метасимвол ? , идущий после .+ , предписывает искать текст до первого символа, указанного следующим. В нашем случае это закрывающая круглая скобка.

Надо запомнить, что в таких запросах ограничивающий метасимвол ? всегда идет последним. С этим метасимволом словесная формулировка запроса звучит так:

«найти открывающую круглую скобку, после которой идут один или несколько символов и первую закрывающую круглую скобку».

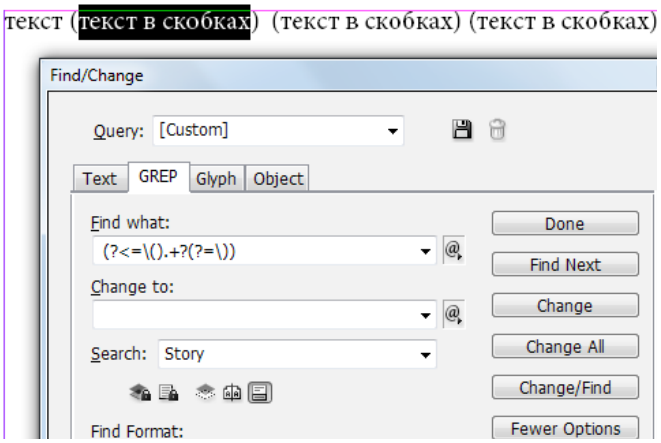


Но как быть, если надо найти текст в скобках, но сами скобки исключить из выделения найденного текста? Очевидно, что в этом случае должен быть применён нежадный поиск.

Для решения этой задачи надо установить условие «Положительный просмотр назад» (Positive Lookbehind) (?<=) , означающее «только если это есть перед текстом», и условие «Положительный просмотр вперёд» (Positive Lookahead) (?=) , означающее «только если это есть после текста».

Запрос может выглядеть так:

(?<=\().+?(?=\))



Как видно, скобки не выделены. Обратите внимание, что перед искомыми скобками стоит обратная наклонная черта \(и \) — метасимвол указывает, что это именно искомые скобки.

Отрицательные просмотры

В GREP-инструментарии есть и Negative Lookbehind (отрицательный просмотр назад) и Negative Lookahead (отрицательный просмотр вперед). В данном случае сложно найти безупречный перевод слова Negative. Пусть будет «отрицательный» как констатация факта, что совпадение ещё не наступило. Словами эти запросы можно сформулировать так: ***«до тех пор, пока это не появится перед текстом»*** и ***«до тех пор, пока это не появится после текста»***.

Позже будут примеры использования этих инструментов.

Упорядочение телефонных номеров с помощью GREP

Так сложилось, что существует очень много вариантов записи телефонных номеров. Попытка упорядочить их — неплохая практика в примерении GREP-инструкций.

Вот список телефонных номеров, с разными разделителями, и без них, со скобками, и без них, в общем, то ещё разнообразие, и его нужно привести к единому формату +7 (XXX) XXX-XX-XX.

Чехарда оформления	Нужно получить
800-123-45-67	+7 (800) 123-45-67
095-123-45-67	+7 (095) 123-45-67
8-812-147-23-65	+7 (812) 147-23-65
1-800-321-98-76	+7 (800) 321-98-76
8 (095)987-65-43	+7 (095) 987-65-43
88121234567	+7 (812) 123-45-67
0951234567	+7 (095) 123-45-67
800.123.45.67	+7 (800) 123-45-67

Десятицифровой телефонный номер — это сам по себе шаблон. Внутри его есть ещё три подшаблона — трёхцифровой код региона в скобках, и одна трёх- и две двухцифровые группы.

Идея в том, чтобы создать такой запрос, который найдёт эти подшаблоны и устранит все возможные неодинаковости в записи номеров. С помощью GREP, используя метасимволы, это будет выполнено одной операцией. Одни метасимволы будут стоять на месте самих чисел, другие будут распознавать объекты текста, которые не всегда могут присутствовать (не появляться ни разу, появляться один раз, появляться несколько раз), третьи будут определять диапазоны цифр.

Ключ к пониманию GREP — думать о поиске не в терминах самого текста (ищется не конкретный текст), а в терминах идеи, как выполнять поиск, что искать (ищется контекст).

Вот формулировка запроса, объясняющая, что должно быть сделано, т.е. идея поиска, и GREP-запрос, решающий эту задачу. Цвет в тексте и запросе поможет понять идею: «*Найти цифру (она может быть, а может и отсутствовать [мб/мо]), сопровождаемую дефисом, точкой или пробелом (они тоже мб/мо), сопровождаемую открывающей круглой скобкой (мб/мо), сопровождаемую группой из трёх цифр, сопровождаемую закрывающей круглой скобкой (мб/мо), затем дефис, точка или пробел (мб/мо), затем группа из четырёх цифр*».

\d?[-.]?\(?(\d{3})\)?[-.]?(\d{3})[-.]?(\d{2})[-.]?(\d{2})

1 2 3 4 5 6 7 8 9 10 11

+7 ($1) $2-$3-$4

а б в г д е ж и к

На картинке выше показан запрос, решающий эту задачу, цифры и буквы, объясняющие каждую операцию. Разберитесь полностью в этом примере, тогда будет проще понимать все остальные запросы. Примеров будет много. Итак, в шаблоне поиска мы обозначаем:

1) совпадение ноль или один раз с одной цифрой перед номером региона;
2) совпадение ноль или один раз с дефисом, точкой или пробелом;
3) совпадение ноль или один раз с открывающей круглой скобкой;
4) заключение в скобки метасимволов создаёт первый подшаблон внутри общего шаблона поиска. Ищутся три последовательных цифры.
5) совпадение ноль или один раз с закрывающей круглой скобкой;
6) совпадение ноль или один раз с дефисом, точкой или пробелом (т.е. это повтор пункта 2);
7) второй подшаблон трёх последовательных цифр в телефонном номере;
8) совпадение ноль или один раз с дефисом или точкой;
9) третий подшаблон двух последовательных цифр в телефонном номере
10) повтор пункта 8;
11) четвертый подшаблон двух последовательных цифр в телефонном номере:

а) код выхода на международную линию;
б) пробел и открывающая скобка;
в) первый подшаблон;
г) закрывающая круглая скобка и пробел;
д) второй подшаблон;
е, и) дефис;
ж) третий подшаблон;
к) четвёртый подшаблон.

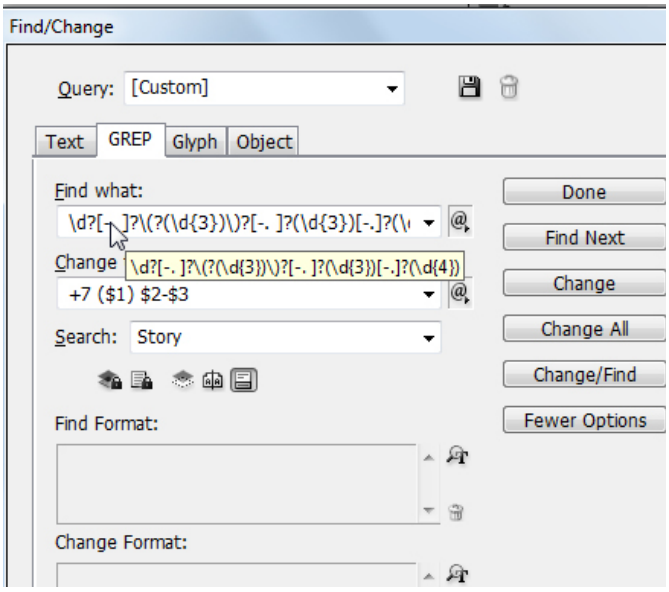
Таким образом, мы привели в порядок телефонные номера, шаблон описания которых выглядит так …-XX-XX, а как быть, если надо упорядочить телефонные номера с идущими в конце подряд четырьмя цифрами цифрами (…-XXXX)?

Тут поможет запрос

Найти: \d?[-.]?\(?(\d{3})\)?[-.]?(\d{3})[-.]?(\d{4})

Заменить: +7 ($1) $2-$3

Содержимое строки поиска часто не помещается полностью в окне, но если поставить на неё курсор, вся строка запроса будет отображена в окне подсказки:



Объединение шаблонов …-XX-XX и …-XXXX в шаблон XXX-XXXX

В исходном тексте телефоны могут быть в любом формате, нам же необходимо оформить их единообразно, допустим, номера они должны быть приведены к шаблону +7 (XXX) XXX-XXXX. С имеющимся багажом знаний мы уже легко это сделаем вот таким понятным запросом:

Н: \d?[-.]?\(?(\d{3})\)?[-.]?(\d{3})[-.]?(\d{2})([-.]?)(\d{2})

З: +7 ($1) $2-$3$5

На этом экспресс-объяснение возможностей GREP можно считать законченным. Если разобраться в приведённых примерах, то уже этих будет знаний достаточно, чтобы понимать простые чужие и готовить свои GREP-запросы. Дальше мы рассмотрим конкретные решения различных случаев, это будет, если хотите, галерея работающих эффективных преобразований.
Как только вы разберётесь в них, у вас появится желание иметь свои галереи таких преобразований.

Удобство чтения строк запросов

Машина читает данные последовательно, а человеку удобно было бы объединять команды в какие-то логические группы. Чтобы пробелы между группами не воспринимались как искомые коды пробелов, перед строкой надо указать команду исключения обработки пробелов, эта инструкция выглядит так: (?x). Тут, правда, можно попасть вот на такие грабли:

Нормально работающая инструкция «Рис. \d+» ищет все слова «Рис.». после которых стоит пробел и номер. Но если перед этой инструкцией поставить (?x), поиск работать не будет. Это произойдёт потому, что пробел перед \d исключается из рассмотрения, поэтому важно всегда для обозначения пробелов ставить \s , а не нажимать на клавишу пробела.

Другой инструмент облегчения работы с запросами — включение в текст комментариев.

Текст комментария помещается между (?# и) и пропускается обработчиком GREP-команд.

Вот комментарий: (?# Инструкции разделены пробелами на группы)'

Поиск пробелов перед переводом строки и удаление их

Если текст сырой, то поиск (\s+)(\r) и замена найденного на $2 решит эту задачу.

Но представьте, что есть строка с абзацным стилем *Стиль 1*, затем строка, у которой *Стиль 2*.

После выполнения приведённого выше запроса строка *Стиль 2* потеряет свой абзацный стиль и получит стилевое оформление предыдущей строки.

Чтобы избежать этого, надо исключить найденный знак перевода абзаца из выделения найденного текста. Это можно сделать положительным просмотром вперед.

(\s+?)(?=\r) , а поле замены пустое, чтобы удалить выделенные пробелы.

Перестановка имён

Предполагается, что в строке есть только фамилия и имя, разделенные одним или несколькими пробелами. Переставить слова можно так:

Н: (\w+)(\s+)(\w+)$

З: $3 \s$1

Здесь второй найденный образец — один или несколько пробелов. Поэтому меняем принудительно на один пробел.

Перестановка имён и удаление возможных пробелов в конце строки

Представьте, вам прислали для беджей файл с сотней-другой информации в таком формате

Фамилия Имя
Должность

Вы сделали файл для печати, отослали на утверждение, и получили ответ, что неужели непонятно, что сперва должно быть имя, а потом фамилия.

Поскольку переданный вам для работы материал зачастую не соотвествует нужному виду, как в данном случае, приходится обрабатывать его самому. И тут может обнаружиться, что в вёрстке после имени иногда встречаются пробелы.

Можно объединить два предыдущих запроса в один, чтобы и от этих пробелов избавиться, и имя с фамилией поменять местами?

Да, вот как это будет выглядеть.

Во-первых, вот такое слепое объединение двух строк «найти», вот такое:

Н: (\w+)(\s+)(\w+)(\s+)(?=\r)

З: $3\s$1

не решит задачу.

Такой запрос будет пропускать «благополучные строки», т.е. те, в которых нет пробела перед знаком перевода строки, из-за того, что операнды

(\s+) ищут один и более пробелов перед знаком перевода строки. Заменим (\s+) на (\s*), т. е. будем искать продел, появляющийся ноль и более раз перед знаком перевода строки. Вот эта формулировка запроса ***ноль и более раз*** учитывает и ситуации, когда перед знаком перевода строки нет пробела!

Н: (\w+)(\s+)(\w+)(\s*)(?=\r)

З: $3\s$1

Обработка дат

Допустим, есть даты в формате dd-mm-yy, dd/mm/yy, dd-mm-yyyy, dd/mm/yyyy, и нужно привести их к одному из двух форматов dd/mm/yy или dd/mm/yyyy, в зависимости от того, как указан год, двумя или четырьмя цифрами. Задача осложняется ещё и тем, что данные могут быть введены в формате d/m/…

В этом примере научимся адресоваться к группе искомых данных в поле поиска. Вот строка поиска: (?x) \d\d? ([/-]) \d\d? \1 \d\d\d\d, для вас в ней уже всё должно быть понятно, кроме \1. Эта инструкция предписывает в этом месте строки поиска подставить первую заключенную в круглые скобки группу параметров. То есть это эквивалентно) (?x) \d\d? [/-] \d\d? [/-] \d\d\d\d, но с сокращенными записями проще работать.

Приведённая выше срока поиска работает исправно, но нам ведь еще необходимо привести найденные данные к определенному.

Поэтому изменим строку поиска, чтобы воспользоваться $-операторами найденного текста.

(?x) (\d\d?) ([/-]) (\d\d?) \2 (\d\d\d\d)

Теперь группа разделителей ([/-]) стала второй заключенной в скобки группой, поэтому и указатель повторяемой группы стал \2.
В строке замены должно быть $1/$3/$4

Сформулированный только что запрос обрабатывает даты, в которых год указан в формате yyyy. Можно сделать, конечно, аналогичный запрос для поиска дат, в которых год в формате yy, но мы с вами уже обладаем знаниями, как сделать один запрос, понимающий yy-формат и yyyy-формат задания года.

(\d\d)? — эта запись найдет две цифры, встречающиеся ноль или один раз. Очевидно, что когда ноль, это формат yy, когда один — это формат yyyy. Поэтому убедитесь, что эта строка работает с обоими форматами года:

Н: (?x) (\d\d?) ([/-]) (\d\d?) \2 ((\d\d)?\d\d)

З: $1/$3/$4

Обратите внимание, что в примерах обработки телефонных номеров, подряд идущие цифры описывались иначе. Это не единственный случай, когда одна ситуация имеет несколько решений.

Поиск текста в скобках

Этот вопрос выше уже обсуждался, и были проверены запросы \(.+\), \(.+?\) и (?<=\().+?(?=\)). И вот пример иного решения задачи, с использованием запроса, работающего значительно быстрее и на бóльших объемах.

Мы знаем, что в квадратные скобки включаются символы одной группы. Кроме того, можно сформировать группу исключения, чтобы при поиске символы этой группы исключались из просмотра.

Вот такой запрос ищет открывающую и закрывающую круглые скобки и текст в них:

\([^)]+\)

\(искомая открывающая круглая скобка;
[начало группы исключения;
^ маркер исключения;
) круглая скобка, она входит в группу исключения;
] закрытие группы исключения;
+ искать один и более символов, пока не будет встречен символ из группы исключения;
\) искомая закрывающая круглая скобка.

Таким образом, в группе исключения все символы, кроме закрывающей круглой скобки. Это значит, что запрос будет искать и выделять все символы, пока не встретит хоть один из символов из группы исключения, в данном случае — закрывающую круглую скобку.

Питер Карел в своих книгах утверждает, что этот вариант поиска работает быстрее, чем запрос \(.+?\), что особенно заметно на больших текстах.

Положительный просмотр вперёд/назад тут тоже работает:

(?<=\()[^)]+(?=\))

можно скопировать, попробовать и убедиться.

Собственно, эта книга как раз для того, чтобы читая пробовать и превращать прочитанное в знания и умения.

Удаление повторов из списка

Другой интересный пример использования группы исключения.

Если есть списки, содержащие идущие подряд одинаковые строки, то запрос:

H: ([^\r]+\r)\1+
3: $1

оставит одну строку из каждой группы одинаковых строк.

Идея этого трюка в следующем: в группе исключения только знак перевода строки. Операнд [^\r]+\r ищет один и более символов, не совпадающих с символом перевода строки, вплоть до первого встреченного знака перевода строки. Этот операнд взят в скобки, поэтому его можно повторить, дав ссылку назад \1+. Поскольку тут есть плюс, то это значит, что данная инструкция предписывает искать одну и более таких строк. В результате будут искаться все *подряд идущие* строки, и они будут *выделены* после нахождения их.

В поле замены стоит $1, т.е. ссылка на шаблон искомой строки, он в поле поиска в круглых скобках, значит, после того как все одинаковые подряд идущие строки будут найдены и выделены, они будут заменены на одну такую строку. Так запрос исключит все повторы из текста.

Обратите внимание, что последняя строка обрабатываемого массива обязательно должна завершаться символом перевода строки. Без него эта строка рассматриваться не будет, и одинаковые строки могут остаться.

Удаление пустых абзацев в начале и в конце статьи

Часто встречается, когда в импортированном тексте есть несколько подряд идущих символов перевода строки в начале и конце статьи. От них можно избавиться вот такими запросами:

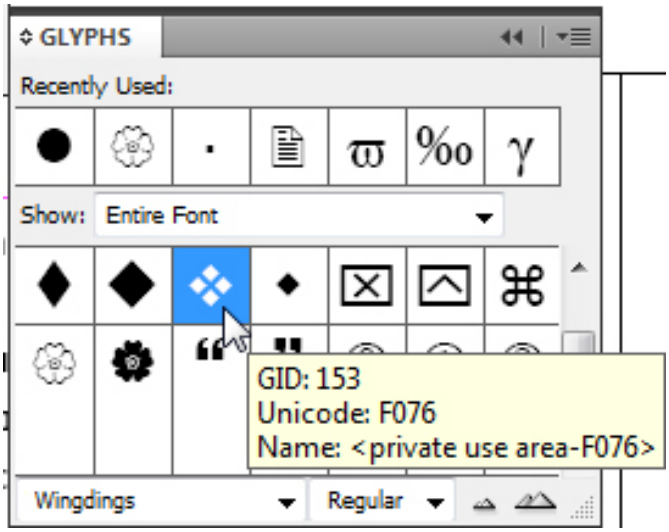
1) в начале статьи: \A\s+
2) в конце статьи \s+\Z ,

в обоих случаях поле замены должно быть пустым.

Добавление символа с конкретным юникодом

Чтобы добавить любой символ из кодовой таблицы любой гарнитуры, надо знать его юникод.

Вот в палитре глифов выбран символ,



его юникод равен F076.

Чтобы добавить его как маркер конца статьи, воспользуйтесь следующим запросом:

H: .\Z
3: $0~y\x{F076}

тут

$0 — найденное, в данном случае конец текста,
~y — табулятор выравнивания по правому краю,
\x{F076} — так, в фигурных скобках, указывается юникод символа. (И обязательно нужно после появления символа в конце текста указать, что это знак из гарнитуры Windings) ❖

Авторы в списке литературы

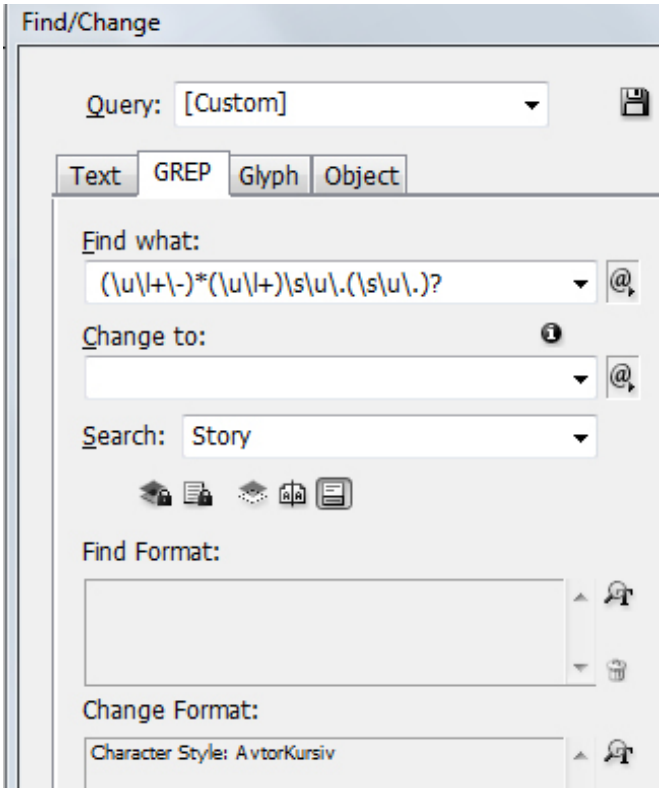
В помещаемом в конце книге списке литературы фамилии авторов обычно выделяют курсивом. Эту задачу можно решить с помощью GREP.

Создайте символьный стиль *AvtorKursiv*, в котором будет определено, что выделенный текст оформляется курсивным начертанием.

Списки авторов составляются по-разному: иногда сразу идут Фамилия И.О., если несколько авторов, то они указываются один за другим. Может быть так, что указываются только фамилия и сокращение имени. Абзацы этого списка могут быть пронумерованы. Очевидно, что запрос

(\u\l+\-)*(\u\l+)\s\u\.(\s\u\.)?

с указанием в поле замены символьного стиля AvtorKursiv решит эту задачу.



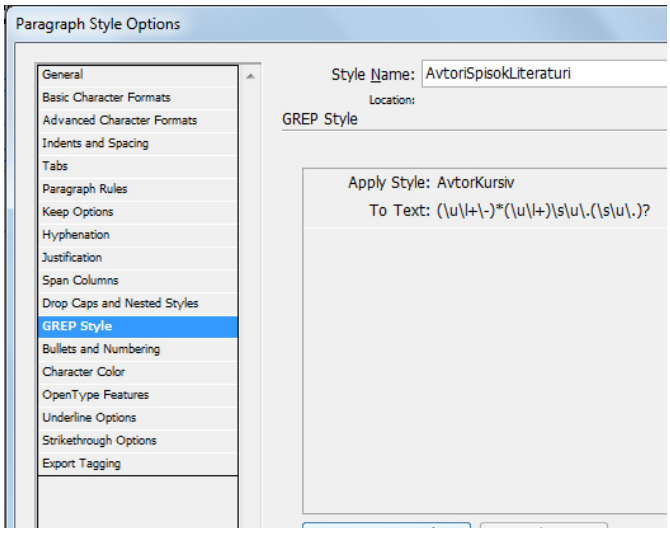
Инструкции GREP в стиле абзаца. Курсив в списке литературы

Итак, мы знаем, что с помощью GREP можно тексту, соответствующему заданному шаблону, присвоить символьный стиль.

А ведь ещё можно встраивать эту процедуру *поиск текста по шаблону и присвоение найденному символьного стиля* в описание абзацного стиля! Это очень мощный инструмент, вот как он работает.

Если вы разобрались с предыдущим примером, то посмотрите, как можно сделать абзацный стиль, автоматически обрабатывающий списки авторов.

На вкладке GREP описания стиля после щелчка на кнопке New GREP Style, в окне Apply Style выберите нужный стиль символа, а в строке To Text: введите шаблон текста.

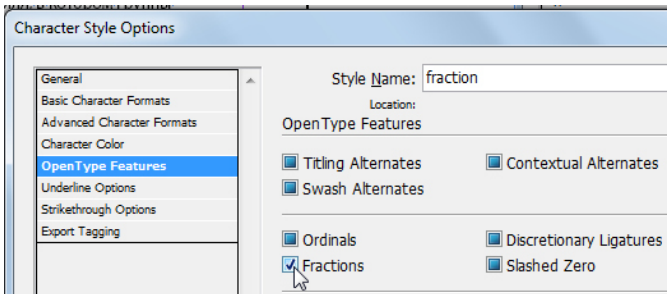


Инструкции GREP в стиле абзаца. Оформление дробей

Другой интересный пример, помогающий понять формулировку GREP в описании абзацного стиля, — создание стиля, в котором группы символов «числитель-наклонная черта-знаменатель» будут оформлены как красивые дроби. Например, обычная дробь 3/4 автоматически превратится в ¾, дробь 7/15 — в ⁷⁄₁₅ и т.п.

Такие преобразования возможны только со шрифтами, имеющими в своём наборе символов варианты написания чисел для числителя и знаменателя. Шрифт Myriad Pro, которым сделана эта книга, отвечает этому условию.

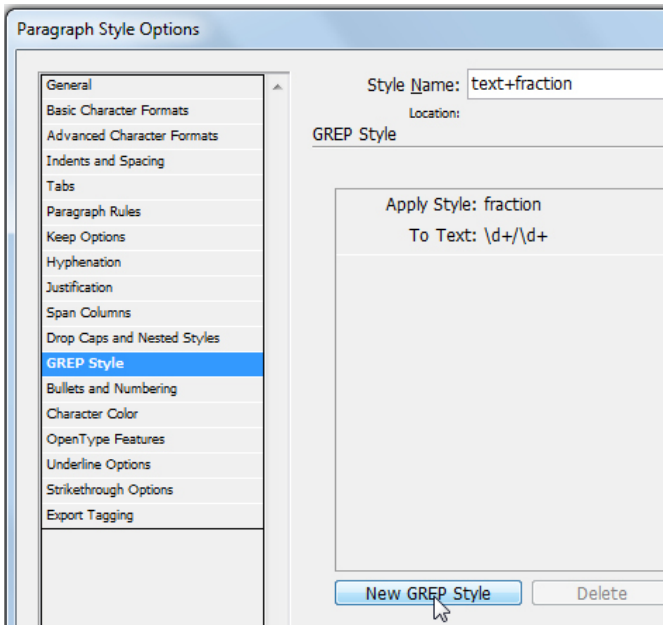
Первое, создадим символьный стиль, назовём его fraction, в котором включим преобразование обычных дробей в красивые (см., где курсор на следующем рисунке).



Дальше всё, как в предыдущем примере: на вкладке GREP выбираем символьный стиль и определяем шаблон, который нужно искать и обрабатывать.

В данном случае шаблон — \d+/\d+ — описывает дробь. Благодаря такому описанию числа будут представляться обычными цифрами,

и только к дробям будет применяться опция преобразования.



Примеры с отрицательными просмотрами

Отрицательный просмотр вперёд — найти все случаи употребления слова "Схема", после которого нет номера.. Это решается так:

(?x) Схема (?!\s\d)

Отрицательный просмотр назад — найти все числа, перед которыми нет знака фунта:

\b(?<!£)\d+

\b — наличие этого оператора обусловливает поиск всего числа, а не его части. Поэкспериментируйте, чтобы понять.

Ограничение отрицательного просмотра

В одной из своих книг Питер Карел, ссылаясь на опыт Джефри Фридла, предостерегает от следующей ошибки оформления записи отрицательного просмотра: *нельзя, чтобы искомые данные были переменной длины*. Например, если в документе есть как правильно оформленные указания на рисунки «Рис.», так и такие, что точка потеряна — «Рис», то верный на первый взгляд поиск этих слов и номера после них (?<=Рис.?\s)\d работать не будет из-за того, что надо в одном запросе искать назад в обрабатываемом тексте слова разной длины, а это трудно реализовать программно. Но для поиска вперёд такого ограничения нет.

Знак доллара и код \x{0024}

Интересно попробовать поработать с последним примером, изменив условие — искать числа, перед которыми отсутствует знак доллара, (ранее в отрицательном просмотре фигурировал знак фунта). На первый взгляд, вот эта строка решит нашу задачу:

\b(?<!\$)\d+

(здесь \$ для того чтобы интерпретатор команд не принял этот знак за указание найти конец статьи). Но это не работает, точнее, работает один раз в абзаце, а это не то, что требуется. Возможно, такой запрос корректно выполняется в других реализациях GREP, но не для версий InDesgn CS6 и младше.

Но есть решение! Надо указать уникод доллара, а не сам знак:

\b(?<!\x{0024})\d+

и запрос найдет все числа, перед которыми не стоит доллар.

Проверка кода

При составлении grep-инструкций обработки текста возможны ошибки. Не стоит переживать и опускать руки, ошибаются все. Главное, что есть люди, копройдя большой путь проб и ошибок, сделали инструменты анализа таких инструкций! Об этих служебных программах подробно написано на сайте *adobeindesign.ru*, нет смысла дублировать здесь всю информацию с ресурса. Вот ссылки, пробуйте, изучайте, работайте:

http://adobeindesign.ru/2010/04/11/rasshifrovka-grep-vyrazhenij/

http://adobeindesign.ru/2012/12/20/what-the-grep-script/

* * * * * *

На этом это короткое знакомство с инструментом GREP завершено. Оно, конечно, не исчерпывающее, но будьте уверены, что информации, представленной здесь, достаточно, чтобы сделать GREP своим инструментом.

А если хочется изучить его досконально, то из всех известных мне изданий, посвященных этой теме, считаю лучшими книги Питера Карела:

GREP in InDesign CS3/4
http://shop.oreilly.com/product/9780596156015.do
Automating InDesign with Regular Expressions
http://shop.oreilly.com/product/9780596529376.do
На мой взгляд, это книги из категории «Должны быть прочитаны, если знаешь английский».

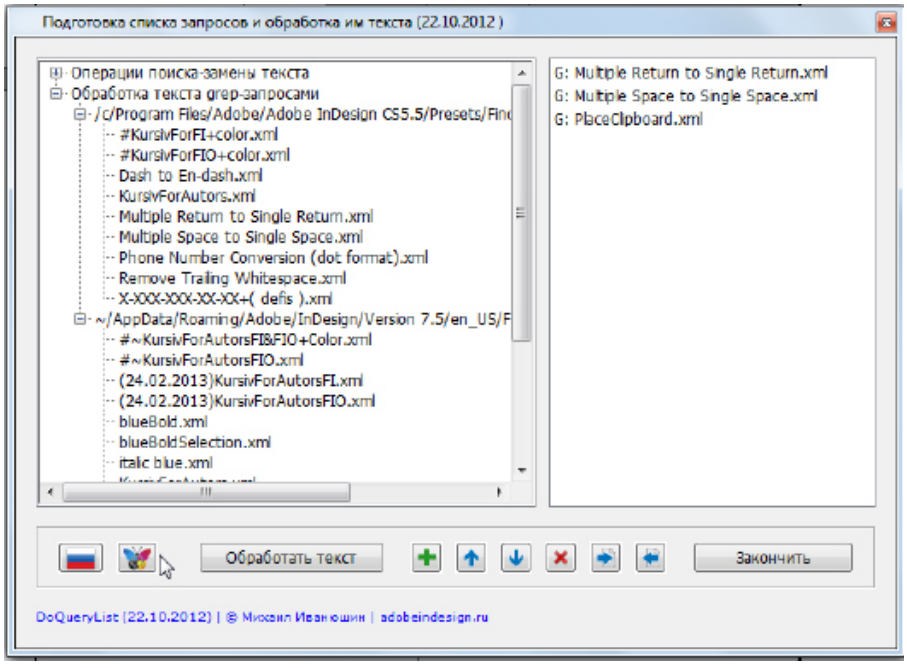
Интересные примеры GREP-запросов вы найдёте на сайте *adobeindesign.ru*, для вывода всех тем в окне поиска введите слово **grep**.

Однако не столь важно, какими руководствами вы будете пользоваться. Важнее, чтобы вы непрерывно совершенствовали свои знания, постоянно пополняли *свою* галерею работающих элегантных преобразований, действительно элегантных, ведь согласитесь, когда GREP встроен в абзацный стиль, и текст обрабатывается в процессе набора, такая изящная возможность радует.

Разбирайтесь, получайте удовольствие от работы с этим замечательным инструментом, и со временем вы обязательно станете в глазах друзей гуру разработки эффективных преобразований.

По мере роста мастерства у вас будут наработки последовательностей GREP-инструкций, выполняющих за несколько шагов различные частные задачи обработки текста. И потенциально возможна проблема, что эти последовательности, просто гирлянды различных эффективных преобразований работают быстро, но много времени тратится на их поиск и запуск.

И тут тоже есть решение — скрипт **DoQueryList**
http://adobeindesign.ru/2012/10/27/doquerylist-programma-obrabotki-teksta-zaprosami/
Идея скрипта — позволить пользователю готовить последовательности GREP-операций, проверять их и сохранять для использования в дальнейшем. С этим скриптом все ваши «гирлянды» будут выполняться в несколько щелчков мышкой. Просто и очень удобно! Эта бесплатная программа возьмёт на себя всю работу по загрузке и выполнению запросов, вам останется только творчество создания их. Зайдите по указанной выше ссылке, подпишитесь, чтобы попробовать её в деле и встроить в ваш рабочий процесс. Она четырёхязычная, и французы уже взяли её в оборот: *http://www.scriptopedia.org/js-indesign/141-doquerylist-fr.html*, и вы не отставайте!
Успехов и удовольствия в грепостроении, ведь это же так прикольно сознавать, что то, что совсем недавно было какой-то бессмыслицей, сейчас стало инструментом, экономящим дни жизни.



GREP — это

Г — грамотно

Р — рационально

Э — эффективно

П — профессионально