Программирование скриптов для Vim. Часть 9. Другие типы плагинов

Предыдущие части

Часть 1. Запуск скриптов
Часть 2. Переменные
Часть 3. Работа со списками
Часть 4. Работа со строками
Часть 5. Операции ветвления и функции
Часть 6. Продвинутое использование функций
Часть 7. Словари и объектно-ориентированное программирование
Часть 8. Более подробно о плагинах

Содержание

Библиотеки скриптов

Иногда бывают ситуации, когда нужно в разных плагинах, которые вы делаете, использовать один и тот же кусок кода. Конечно, можно его просто копировать из одного плагина в другой, но лучше всего оформить его в виде функции, находящейся в библиотеке, тогда при нахождении ошибки в этом участке кода не придется искать где этот же кусок кода находится в другом плагине, а будет достаточно исправить функцию в библиотеке.

Студентам

Фото

- Животные
- Черно-белые
- Пейзажи/Природа
- Город
- Закаты
- Панорамы
- Спорт
- Репортаж
- Разное

Контакты

Библиотеки могут быть полезны, если вам в разных скриптах потребуется, например, функция для чтения файлов в определенном формате, или функция для особой обработки текста. В основном функции из таких библиотек используются внутри других скриптов, и они не предназначены для использования их конечным пользователем.

Для организации такой библиотеки скриптов служат директории *autoload*, расположенные, как и директории *plugin*, в нескольких местах: в корневой директории Vim, в поддиректории *after* и в домашней директории пользователя. Подробно расположение директорий плагинов мы разбирали в прошлой части.

Для создания библиотеки скриптов нужно соблюдать следующие требования к именам функций:

Если функция *Foo()* содержится в файле *autoload/bar.vim*, то объявление функции должно выглядеть следующим образом (обратите внимание на имя):

```
function! bar#Foo()
    ...
endfunction
```

Исходник

Имя *Foo()* должно начинаться с заглавной буквы, так как эта функция будет глобальной, а перед именем функции должно располагаться имя файла *bar* (без расширения .vim), которое отделяется решеткой #.

При вызове этой функции также необходимо указывать ее имя в таком же виде:

```
call bar#Foo()
```

Исходник

Разработчики Vim на этом не остановились и сделали возможность хранить скрипты в поддиректориях, в этом случае и имена функций тоже должны содержать имена поддиректорий, разделяя их вместо символа "/" символом решетки "#". Например, если функция *Foo()* расположена в скрипте, который, в свою очередь, расположен в директории *autoload/myscripts/cool/bar.vim*, то функция должна быть объявлена следующим образом:

```
function myscripts#cool#bar#Foo()
    ...
endfunction
```

Исходник

Именно с таким именем ее и придется вызывать командой *call*:

```
call myscripts#cool#bar#Foo()
```

Исходник

Давайте для примера напишем Hello Word из библиотечной функции. Создадим файл *hello.vim* в директории *autoload/myscripts/cool*. В качестве содержимого этого файла будет следующий простой код:

```
function! myscripts#cool#hello#Hello(name)
    return "Hello, " . a:name
endfunction
```

Исходник

Теперь после перезапуска редактора мы можем выполнить следующую команду:

```
:echo myscripts#cool#hello#Hello("Vim")
```

Исходник

В результате получим вполне ожидаемое:

```
Hello, Vim
```

Плагины для определенных типов файлов

Определение типов файлов

В Vim есть возможность создавать плагины, которые будут работать только с определенными типами файлов. Для начала давайте разберемся в том что из себя представляют эти самые типы файлов. Типы файлов и расширения файлов - это не одно и то же, хотя и есть возможность задать как должен называться тип файла для определенных расширений. Например, если открыть файл с расширением .py или .pyw, то тип файла будет называться "python".

Самый примитивный способ задания типа файла состоит в том, чтобы для уже открытого файла выполнить следующую

команду

```
:set filetype=mytype
```

Исходник

Теперь открытый файл будет иметь тип "mytype" до следующего открытия этого файла. При этом, если открытый файл до этого использовал раскраску синтаксиса, то она будет отключена, так как для типа "mytype" плагин синтаксиса не установлен. Если аналогичным образом мы вернем файлу его старый тип, для которого раскраска синтаксиса установлена, то текст опять станет цветным. Про плагины для раскраски синтаксиса мы поговорим чуть позже в этой статье.

Каждый раз при открытии файлов задавать их тип вручную не удобно, поэтому в Vim существует возможность автоматически их определять. Для этого можно использовать команду *autocmd*, или в сокращенном виде - *au*. Эта команда в упрощенном виде имеет следующий синтаксис:

```
:autocmd {event} {pat} {cmd}
```

Здесь пропущены два необязательных параметра, которыми мы пользоваться не будем, для более подробной информации об этой команде выполните команду *:help au*.

Эта команда работает следующим образом. Если возникает событие, перечисленное в параметре {event} (несколько событий перечисляются через запятую), и при этом событие относится к файлу, имя которого совпадает с шаблоном {pat}, то вызывается команда {cmd}. Полный список событий можно найти в документации, если ввести команду *:help autocmd-events*. Для примера, существуют события, возникающие при открытии файла или при создании нового файла (именно их мы и будем использовать), события, возникающие при записи файла на диск, при операциях с буферами (создание, удаление, переключение) и много других событий.

В качестве шаблона файлов {pat} можно использовать как обычные файловые шаблоны наподобие *.txt, так и регулярные выражения.

В качестве {cmd} может выступать как простая команда, так и вызов функции, которая должна быть предварительно объявлена.

Давайте создадим правило, по которому будет создаваться новый тип файлов. Пусть это будет тип "pmwiki" для файлов с маской "*.pmwiki" (как я уже не один раз говорил в прошлых частях, pmwiki - это вики-нотация, в которой я набираю эту статью). Для осуществления нашей задумки нам достаточно добавить в файл _vimrc/.vimrc следующую строку:

```
autocmd BufRead,BufNewFile *.pmwiki set filetype=pmwiki
```

Исходник

Этой командой мы говорим, что при возникновении события открытия (BufRead) или создания (BufNewFile) файла, который удовлетворяет маске *.pmwiki, должна вызываться команда *set filetype=pmwiki*, которая и устанавливает тип файла как "pmwiki".

Поделюсь еще одной полезной настройкой, которой я пользуюсь. При сохранении текущей сессии в Vim с помощью команды *mksession* (открытые файлы, расположение буферов на экране и т.п.) в качестве имени файла с сессией я задаю расширение .session. А в файле настроек _vimrc/.vimrc у меня прописана следующая строка:

```
autocmd BufRead *.session so %
```

Исходник

Благодаря этому как только в Vim открывается файл с таким расширением, вызывается команда выполнения этого файла как скрипта ('so %', или в полном виде *source %*), что приводит к автоматическому восстановлению сессии.

ftplugin

Ну что ж, пришло время заняться непосредственно плагинами. Плагины, которые должны работать только с файлами определенных типов, располагаются в директории *ftplugin*. При их написании нужно соблюдать некоторые правила.

Во-первых, должно выполняться одно из следующих трех условий:

- Имя файла плагина (без учета расширения .vim) должно совпадать и именем типа файла, для которого плагин должен работать. Например, плагин для типа "pmwiki" должен так и называться *pmwiki.vim* и располагаться непосредственно в папке *ftplugin*.

- Имя файла плагина (без учета расширения .vim) должно начинаться с имени типа файла, а затем после символа подчеркивания "_" могут идти любые символы. Этот способ именования файлов плагинов удобен, если необходимо загружать несколько плагинов для одного и того же типа. Например, файл может называться *pmwiki_plugname.vim* и располагаться непосредственно в папке *ftplugin*.

- Имя файла плагина может быть любым, но сам файл должен располагаться в поддиректории, имя которой совпадает с именем типа файла. Например, файл может называться *plugname.vim* и располагаться в папке *ftplugin/pmwiki/*.

Второе правило написания таких плагинов относится к исходному тексту плагина. Так же как и для обычных плагинов, которые мы рассматривали в прошлой части, мы должны предоставить пользователю возможность не загружать наш плагин без удаления самого файла плагина.

Это самое второе правило разработчики плагинов понимают по-разному. Дело в том, что в документации написано, что необходимо вставить следующий код в самое начало исходника плагина:

```
" Only do this when not done yet for this buffer
if exists("b:did_ftplugin")
    finish
endif
let b:did_ftplugin = 1
```

Исходник

Тогда пользователь сможет заблокировать загрузку плагина, добавив в свой файл .vimrc/_vimrc следующую команду:

```
let b:did_ftplugin = 1
```

Исходник

Но если пользователь добавит эту строку в свой файл настроек, то будут заблокированы все плагины, использующие переменную *b:did_ftplugin*, поэтому некоторые разработчики используют другое имя переменной, чтобы пользователь мог блокировать именно их плагин, а не все. Надо сказать, что плагины, которые прилагаются к Vim по умолчанию, написаны таким образом, что используют именно переменную с именем *b:did_ftplugin*, но мне лично больше нравится, когда используют другое имя, так как в этом случае у пользователя появляется больше возможностей для настройки своего Vim.

В завершение этого раздела давайте модернизируем плагин, который мы создавали в конце пятой части. Напомню, что там мы делали плагин, который создавал оглавление для статей, набранных в нотации pmWiki. Так как теперь у нас для pmWiki есть свой тип файлов, то сделаем так, чтобы плагин работал только для буферов, в которых открыт этот тип файлов.

Не будем повторно разбирать то как работает этот плагин, нас в данный момент интересует его начало, чтобы иметь наглядное представление с чего начинаются подобные плагины.

```
" Function for pmWiki (ftplugin)
" Last Change:  2009 Sen 19
" Maintainer:   Jenyay <jenyay.ilin@gmail.com>
" License:      This file is placed in the public domain.

if exists("b:did_pmwiki_ftplugin")
    finish
endif
let b:did_pmwiki_ftplugin = 1

function! s:PmTitles()
    " Регулярное выражение для нахождения якоря
    let l:pattern_anchor = '\m^\[\[#\([a-zA-Z0-9_]*\)\]\]'

    " Регулярное выражение для нахождения заголовка
    let l:pattern_title = '\m^!!\s*\(.*\)$'

    " Количество строк в файле
    let l:count = line ("$")

    " Сюда будут добавляться строки оглавления
    let l:result = []

    " Проходимся по всем строкам в буфере.
    for n in range (1, l:count)
        " Получим номер строки по ее номеру
        " Строки в буфере нумеруются с 1
        let l:currline = getline (n)

        " Для каждой строки проверим соответствует ли она
шаблону с якорем
        let l:anchor = matchlist (l:currline,
```

```
l:pattern_anchor)

        if len (l:anchor) != 0
            " Если строка соответствует, то список не будет
пустым

            " Теперь проверим соответствует ли следующая
строка шаблону заголовка
            let l:nextline = getline (n + 1)
            let l:title = matchlist (l:nextline,
l:pattern_title)

            if len (l:title) != 0
                " Если и заголовок найден, создадим строку
оглавления
                let l:resline = printf ("* [[#%s | %s]]",
l:anchor[1], l:title[1])
                call add (l:result, l:resline)
            endif
        endif

    endfor

    " Добавим ссылку на комменатрии
    call add (l:result, "* [[#comments | Комментарии]]")

    " Получить положение курсора в виде списка:
    " [номер буфера,
    " номер строки,
    " номер столбца,
    " параметр при использовании опции virtualedit]
    let l:cursor = getpos(".")

    " Вставим полученное оглавление в ту строку, где сейчас
стоит курсор
    call append (l:cursor[1], l:result)

    " Удалим все переменные
    unlet l:pattern_anchor l:pattern_title l:count
    unlet! l:resline l:nextline l:title l:anchor l:currline
endfunction

command PmTitles call s:PmTitles()
```

Исходник

Кроме проверки на необходимость загружать сам плагин в самое начало добавлены комментарии с информацией о плагине. Кроме этого было бы неплохо еще добавить файл с документацией (как это делать было написано в прошлой статье), но в данный момент мы не будем этим заниматься.

Раскраска синтаксиса

Еще один тип плагинов, которые работают для определенных типов файлов, являются плагины синтаксиса. Именно благодаря им Vim раскрашивает ключевые слова, операторы и другие элементы в исходных текстах программ. Разумеется, что мы можем создавать свои плагины синтаксиса, которые могут сделать более удобной работу с теми файлами, для редактирования которых используется Vim. Например, это могут быть файлы, в которых используются редкая нотация (да все та же pmWiki), или это могут быть обычные текстовые файлы, в которых вы хотели бы выделять отдельные элементы.

Тема создания плагинов синтаксиса, вообще говоря, довольно большая, поэтому мы рассмотрим только самые основы.

Процесс создания файла синтаксиса можно разделить на два этапа: сначала определяются элементы синтаксиса, которые должны быть раскрашены, а затем определяется то, как они будут раскрашиваться. Первый этап выполняется с помощью команды *syntax*, а второй с помощью команды *highlight*.

Несмотря на то, что используется всего две команды, количество возможных параметров у них достигло такого количества, что иногда одну и ту же команду при разных наборах параметров удобнее рассматривать как разные команды.

В этом разделе мы будем создавать файл синтаксиса для раскраски текста в нотации pmWiki.

Выделение элементов синтаксиса

Первое, что нам нужно сделать, это создать группу элементов, которые будут раскрашиваться однотипно. Типичным случаем таких элементов являются ключевые слова, комментарии, строковые константы и тому подобные элементы.

Для начала создадим в папке *syntax* пустой файл с именем *pmwiki.vim*.

Давайте сделаем так, чтобы к тексте pmWiki раскрашивались ключевые слова, заголовки и ссылки, а текст между тремя одинарными кавычками выделялся полужирным шрифтом. Для этого сначала воспользуемся командой *syntax* в следующем виде:

```
syntax [{настройки}] {тип} {группа} {определение} [{настройки}]
```

Здесь {тип} - это один из трех возможных типов синтаксиса:

- *keyword*. В этом случае {определение} - это просто перечисление ключевых слов через пробел. По

умолчанию ключевые слова регистрозависимы. Максимальная длина ключевого слова - 80 символов

- *match*. В этом случае {определение} задается в виде регулярного выражения, заключенного в символы прямого слеша: /.../.

- *region*. Команда раскрашивает текст между заданными открывающимися и закрывающимися символами. Типичным примером для использования такого типа синтаксиса являются комментарии в C-подобных языках, когда нужно пометить текст между /* и */ как комментарий.

Более подробно работу со всеми этими типами синтаксиса мы еще разберем на примерах.

Необязательные параметры {настройки} могут быть заданы как в начале, непосредственно после команды *syntax*, в самом конце или даже между перечислением ключевых слов в {определении}.

Параметр {группа} определяет имя группы, к которым относится элемент синтаксиса, определяемый командой. Так как разработчики Vim старались унифицировать раскраску синтаксиса, чтобы ключевые слова для разных языков программирования раскрашивались одинаково, то большое количество групп уже созданы, и их раскраска уже задана в плагине тем, которые находятся в директории *colors*. Если элемент синтаксиса не попадает в уже существующую группу, то раскраска этого элемента становится сложнее, так как затем надо будет указать как будет раскрашиваться новая группа.

К стандартным относятся следующие группы:

- *Comment*
- *Constant*
- *Identifier*
- *Statement*
- *PrePro*
- *Type*
- *Special*
- *Underlined*
- *Ignore*

- *Error*
- *Todo*

Внутри почти каждой из этих групп (кроме *Comment*, *Underlined*, *Ignore*, *Error* и *Todo*) существуют еще и подгруппы, которые позволяют более гибко настраивать раскраску. Например, внутри группы *Type*, предназначенной для раскраски типов переменных, есть подгруппы *StorageClass*, *Structure* и *Typedef*.

Список всех стандартных групп с их подгруппами вы можете увидеть, если наберете в командной строке Vim команду *:help group-name*. Кроме того, мы можем увидеть все существующие группы (не только стандартные), если введем команду *:highlight* (или в сокращенном виде просто *:hi*).

Создание раскраски pmWiki

Давайте, наконец, начнем заполнять файл раскраски *pmwiki.vim*. Первое, что мы сделаем - это научим Vim выделять ключевые слова, и начнем мы с ключевого слова *Attach*, которое создает ссылку на файл, прикрепленный к странице. Добавим в файл *pmwiki.vim* следующие строки:

```
if exists("b:current_syntax")
    finish
endif

let b:current_syntax="pmwiki"

syntax keyword Keyword Attach
```

Исходник

В самом начале файла мы определяем переменную *b:current_syntax*, это требование ко всем плагинам синтаксиса.

На последней строке мы создаем ключевое слово *Attach*, раскраска которого теперь будет соответствовать раскраске остальных ключевых слов. На следующем скриншоте вы можете увидеть как это слово раскрашивается при моих настройках:

```
1
2 Это пример файла, прикрепленного к странице - Attach:screenshot.png
```

Для первого раза неплохо, однако такое применение команды *syntax* заставит Vim подкрашивать слово *Attach*

даже в тексте (кстати, ключевые слова по умолчанию регистрозависимы).

```
1
2 Это пример файла, прикрепленного к странице - Attach:screenshot.png
3 А это слово Attach в тексте
```

Было бы лучше, если бы раскрашивалось только слово *Attach:* с двоеточием, однако ключевые слова могут содержать только те символы, которые заданы параметром *iskeyword* (или сокращенно *isk*). По умолчанию для Windows значение этого параметра равно "@,48-57,_,128-167,224-235", а для остальных систем - "@,48-57,_,192-255". Первой идеей было, добавить в *iskeyword* еще и знак двоеточия, а затем определить синтаксис как *syntax keyword Keyword Attach:*, однако у меня это не сработало.

Для решения этой проблемы существует другой способ с использованием типа синтаксиса *match*:

```
if exists("b:current_syntax")
    finish
endif

let b:current_syntax="pmwiki"

syntax match Keyword /Attach:/
```

Исходник

Здесь для группы *Keyword* выделяются элементы синтаксиса, которые соответствуют регулярному выражению *Attach:* (без прямых слешей). Теперь раскрашенный текст будет выглядеть следующим образом:

```
1
2 Это пример файла, прикрепленного к странице - Attach:screenshot.png
3 А это слово Attach в тексте
```

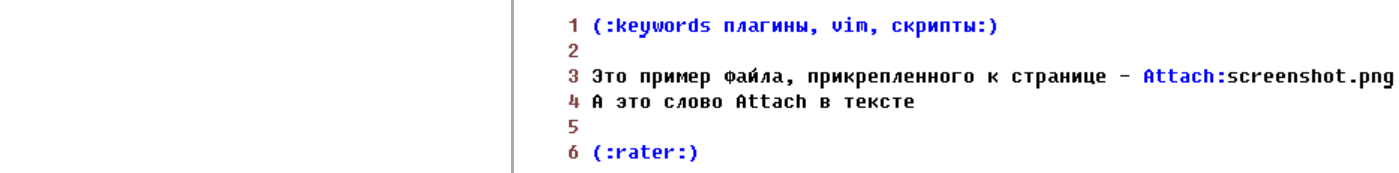
В нотации pmWiki многие команды задаются между символами (: ... :), давайте научим Vim выделять и их. Для этого можно использовать тот же тип синтаксиса *match*, но проще всего использовать тип *region*, в котором удобно задавать начало и конец выделяемой области. Добавим в конец файла синтаксиса новую строку, содержимое файла теперь выглядит следующим образом:

```
if exists("b:current_syntax")
    finish
endif

let b:current_syntax="pmwiki"

syntax match Keyword /Attach:/
```

```
syntax region Statement start=/(:/ end=/:)/
```

Исходник

Здесь мы используем группу *Statement*, для которого *Keyword* является подгруппой. В файле темы Vim, которая у меня используется *Statement* и *Keyword* раскрашиваются одинаково, в результате раскрашенный текст будет выглядеть следующим образом:



Главное отличие типа *match* от *region* в том, что синтаксические конструкции, определенные с помощью типа *match* не могут разрываться переносом строки, а элементы синтаксиса *region* могут быть многострочными.

Давайте добавим еще два элемента синтаксиса в наш плагин раскраски:

```
if exists("b:current_syntax")
    finish
endif

let b:current_syntax="pmwiki"

syntax match Keyword /Attach:/

syntax region Statement start=/(:/ end=/:)/

syntax match Underlined /\M[[\.\{-}]]/

syntax match Statement /%\w*%/
```

Исходник

Предпоследняя строка делает подчеркнутыми ссылки, которые задаются в двойных квадратных скобках. Здесь для наглядности мы используем немагический режим регулярных выражений (определяемый строкой \M), чтобы не нужно было бы добавлять обратный слеш перед каждым символом квадратных скобок. Также здесь используется выражение *{ - }*, которое работает так же, как и * с той разницей, что * старается найти наибольший участок текста, совпадающий с регулярным выражением (жадный алгоритм), а *{-}* ищет наименьший участок текста (нежадный алгоритм).

Последняя строка выделяет команды, заключенные между символами процента.

Теперь раскрашенный текст будет выглядеть следующим образом:

```
1 (:keywords плагины, vim, скрипты:)
2
3 Это пример файла, прикрепленного к странице - Attach:screenshot.png
4 А это слово Attach в тексте
5
6 (:rater:)
7
8 Это якорь в тексте: [[#anchor]], а это ссылка на другую страницу: [[Programming/Vim]]
9
10 %center%Центрированный текст
```

Использование нестандартных групп синтаксиса

До сих пор мы обходились только стандартными группами для определения синтаксиса, однако часто требуется больше видов раскраски, чем предоставляется по умолчанию. В этом случае в команде *syntax* мы просто задаем имя группы, которую хотим создать, а затем необходимо указать с помощью команды *highlight* (или сокращенно *hi*) каким образом эта группа должна быть раскрашена.

Общий формат команды *highlight* следующий:

```
highlight {группа} {параметры}
```

Здесь {группа} - название группы, определенной с помощью оператора *syntax* или имя стандартной группы.

С помощью команды *highlight* можно указывать разные цвета для графического и консольного режима Vim, для темы с темным фоном и светлым. У команды *highlight* довольно много параметров, поэтому перечислим только некоторые из них.

- *term*. Определяет начертание шрифта (курсив, полужирный и т.п.). Значением этого параметра должен быть список констант, перечисленных через запятую. Наиболее часто используемые возможные значения следующие: *bold*, *italic*, *underline*. Этот параметр будет учитываться на обычных черно-белых терминалах.

- *cterm*. То же, что и *term*, но будет использоваться на цветных терминалах.

- *ctermfg* - цвет надписей для группы на цветном терминале.

- *ctermbg* - цвет фона для группы на цветном терминале. Список именованных цветов для последних двух параметров вы можете узнать, если наберете команду *:help cterm-colors*.

- *guifg* - цвет надписей для группы в графическом Vim (gVim).

- *guibg* - цвет фона для группы в графическом Vim (gVim). Список цветов для gVim вы можете узнать, если наберете команду *:help gui-colors*. Также здесь можно использовать цвета, определенные в формате #rrggbb (#ff00ff).

- *font* - шрифт, который будет использоваться при отображении группы в графическом Vim (gVim).

Давайте теперь создадим несколько нестандартных групп

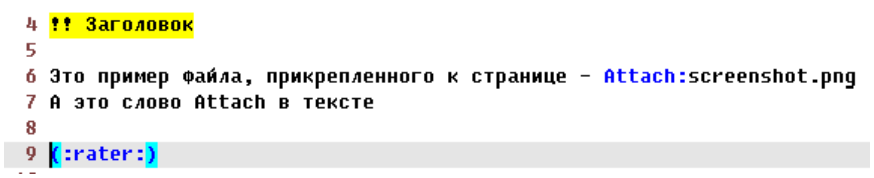
Сначала сделаем так, чтобы Vim выделял заголовки, которые обозначаются двумя или большим количеством восклицательных знаков, которые стоят в начале строки. Заголовки будут выделяться желтым фоном. Для этого нам нужно добавить всего две строки:

```
syntax match header /^!!\+.\+/
highlight header ctermbg=Yellow guibg=Yellow
```

Исходник

В первой строки мы создаем группу *header*, которая должна соответствовать написанному регулярному выражению. На второй строке мы указываем как эта группа должна раскрашиваться. Здесь мы указали, что в цветном терминале и в gVim должен быть желтый фон. На черно-белом терминале эта группа выделяться не будет.

На следующем скриншоте видно как будет выглядеть заголовок:



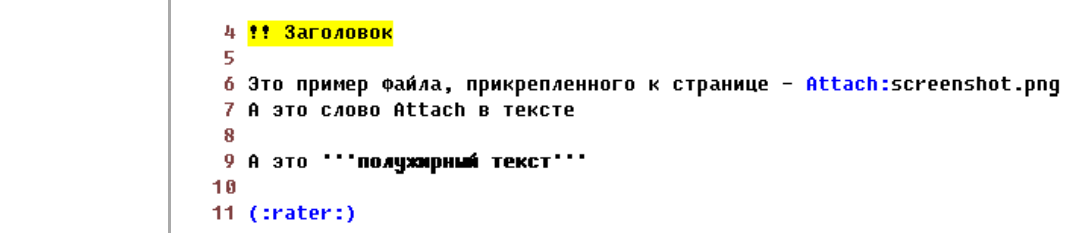
Следующий пример показывает как можно сделать полужирное выделение текста, размещенного между тройными одинарными

кавычками:

```
syntax region bold start=/'''/ end=/'''/
highlight bold term=bold cterm=bold gui=bold
```

Исходник

В результате мы получим следующий внешний вид:

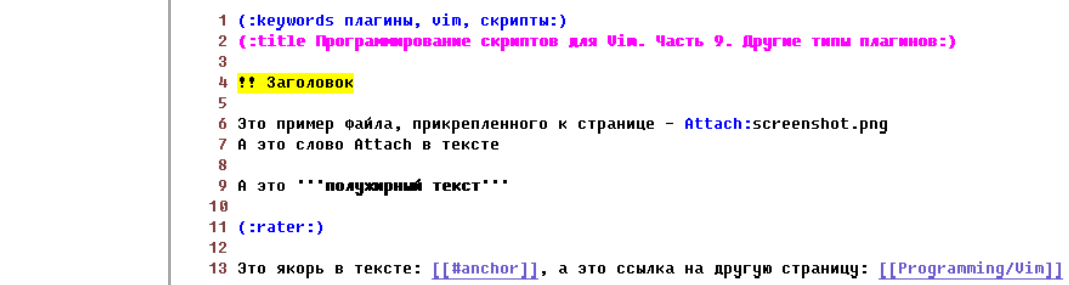


Чуть выше мы уже определили раскраску для команд между скобками (: ... :), однако, если мы хотим какие-нибудь команды выделить отдельно, то достаточно определить новую раскраску, которая по сути будет являться частным случаем другой группы. Для примера сделаем так, чтобы команда (: title :) выделялась полужирным шрифтом и пурпурным цветом:

```
syntax region titleCommand start=/(:title/ end=/:)/
highlight titleCommand ctermfg=Magenta guifg=Magenta
cterm=bold term=bold gui=bold
```

Исходник

Теперь текст в нотации pmWiki, использующий все раскрашиваемые элементы, будет выглядеть следующим образом:



Полностью файл раскраски теперь выглядит следующим образом:

```
if exists("b:current_syntax")
    finish
endif

let b:current_syntax="pmwiki"

syntax match Keyword /Attach:/

syntax region Statement start=/(:/ end=/:)/
```

```
syntax match Underlined /\M[[\.\{-}]]/

syntax match Statement /%\w*%/

syntax match header /^!!\+.\+/
highlight header ctermbg=Yellow guibg=Yellow

syntax region bold start=/'''/ end=/'''/
highlight bold term=bold cterm=bold gui=bold

syntax region titleCommand start=/(:title/ end=/:)/
highlight titleCommand ctermfg=Magenta guifg=Magenta
cterm=bold term=bold gui=bold
```

Исходник

Формально текст с командой *title* попадает под две группы: *titleCommand* и *Statement*, однако из-за того, что группа *titleCommand* объявлена ниже, то приоритет отдается именно ей.

На этом давайте пока прервемся, тема создания плагинов синтаксиса довольно объемная и ей можно посвятить целую статью, а то и не одну.

Как всегда, жду ваших комментариев.

Вы можете подписаться на новости сайта через RSS, Группу Вконтакте или Канал в Telegram.

★★★★★

Рейтинг 5.0/5. Всего 52 голос(а, ов)

○Плохо ○Так себе ○Неплохо ○Хорошо ○Отлично

Голосовать

~ 23.10.2009 - 15:25

~

Первыйнах!

Александр 03.12.2009 - 15:25

Спасибо за серию.

Прочитал и сделал ф-ии, которые выгружают и загружают из MS SQL исходники процедур и представлений.

Jenyay 03.12.2009 - 21:54

Александр, и Вам спасибо. Приятно, когда то, что делаешь, кому-то оказывается полезным :)

Андрей 12.01.2010 - 16:47

Спасибо за статьи, узнал про вим много для себя нового. Надеюсь, будут продолжения :)

Jenyay 12.01.2010 - 17:01
Андрей, Вам тоже спасибо. По поводу продолжения есть идея как минимум для еще одной статьи, но когда до нее доберусь пока не знаю.

Сергей 16.02.2010 - 01:50
Спасибо за серию отличных статей.

West Wind 14.07.2010 - 23:34
Очень познавательно :)
Пытаюсь перелезть на VIM... статьи очень содержательные.

Andrey 19.07.2010 - 22:11
Присоединяюсь ко всем выше...
Статьи замечательные.
Ждёмс продолжения)))

Al 16.08.2010 - 15:02
Спасибо за серию. Уже год на VIM, но так руки не доходили до написания скриптов. Много стало на свои места. Жду еще статей.

Ник 05.10.2010 - 14:52
Спасибо большое, было очень интересно.

Yurgen 26.05.2011 - 22:57
syntax
Статьи действительно просто прекрасные. Огромное спасибо.
В вопросе подсветки синтаксиса хочется разобраться более детально. Например:
syntax region String start=/"/ end=/"/
syntax region Comment start=+/*+ end=+*/+

В этом случае текст в кавычках будет выделяться даже в комментариях, в не зависимости от порядка указания. Интересно как можно указать зависимости одних групп относительно других. Например что-бы в комментариях ничего не выделялось.

З.Ы В английском не силен почитать, а на русском детального описания не нашел. Пробовал играться с контейнерами и кластерами, но запутался только больше.

klay 31.03.2012 - 22:59
очень здорово

супер вообще. весь цикл статей

Vite4eg 31.05.2016 - 13:34
Шикарно
Очень полезный цикл. Спасибо Вам большое!

Ярослав 28.11.2017 - 14:18
Может добавите материала и книгу выпустите? Только материал с сайта не удаляйте. Очень полезно.

Подписаться на комментарии

Автор:

Тема:

Ваш комментарий

Введите код
901

Послать