

jenyay.net

Софт, исходники и фото

Поиск:

>>

[Домой](#) [Блог](#) [Контакты](#)[Печать](#) [Править](#)

Блог

Программки

OutWiker (rus)[Плагины](#)[Бета-версии](#)[Локализации](#)[Документация](#)[Предложения и](#)[баги](#)[Исходники](#)**OutWiker (en)**[Plug-ins](#)[Beta versions](#)[Translate](#)[Suggestions and](#)[bugs](#)[Source code](#)[Documentation](#)**Другие...****Программирование**[Python](#)[Rust](#)[.NET/C#](#)[C++](#)[PHP](#)[Алгоритмы](#)[Инструменты](#)[Остальное](#)**Обзоры книг**

Программирование скриптов для Vim. Часть 8. Более подробно о плагинах

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](#)

Оглавление

- [Структура директорий в Vim](#)
- [Оператор command](#)
- [Комментарии](#)

Итак, мы подошли к самому интересному, а именно к написанию плагинов. В конце [пятой части](#) мы уже писали скрипты, которые можно назвать плагинами, так как они автоматически загружались из папки *plugin*, но там мы не соблюдали некоторые условные договоренности, которые должны соблюдать те разработчики, которые хотят распространять свои творения. Эти договоренности, в конечном итоге, должны помочь пользователям не запутаться в огромном количестве скриптов, существующих для Vim, возможность следить за обновлениями, а иногда и избежать конфликта между разными плагинами.

Студентам

Фото

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

Контакты

Структура директорий в Vim

Для начала разберем структуру папок Vim, из которых он автоматически загружает скрипты, и рассмотрим возможные типы плагинов. На самом деле существует несколько директорий, где могут располагаться плагины. Список этих директорий можно менять с помощью параметра *runtimepath* (или сокращенный вариант *rtp*) в файле *.vimrc* / *_vimrc*. Этот параметр хранит список путей через запятую, где располагаются поддиректории для плагинов различных типов (про типы плагинов поговорим чуть позже). Порядок, в котором расположены пути, также являются порядком, в котором Vim просматривает папки и загружает скрипты из них. Ниже приводятся значения параметра *runtimepath* по умолчанию для Unix и Windows (взято из справки):

```
Unix: "$HOME/.vim,  
$VIM/vimfiles,  
$VIMRUNTIME,  
$VIM/vimfiles/after,  
$HOME/.vim/after"  
Windows: "$HOME/vimfiles,  
$VIM/vimfiles,  
$VIMRUNTIME,  
$VIM/vimfiles/after,  
$HOME/vimfiles/after"
```

- *\$HOME* - это путь до рабочей директории текущего пользователя (например, под Windows XP это *X:\Documents and Settings\USERNAME*).
- *\$VIM* - корневая папка для Vim (например, у меня это *C:\Program Files\Vim*).
- *\$VIMRUNTIME* - путь до запускаемого файла Vim (у меня это *C:\Program Files\Vim\vim72*)

Вы можете посмотреть порядок обхода папок на вашем компьютере, если наберете следующую команду в командной строке Vim:

```
echo &runtimepath
```

[Исходник](#)

Здесь знак "&" перед *runtimepath* обозначает, что это не простая переменная, а параметр самого Vim. Например,

если я выполню эту команду в своем профиле под Windows, то увижу следующие пути (для наглядности я одну длинную строку я разбил на несколько)

```
C:\Documents and Settings\Jenyay\vimfiles,  
C:\Program Files\Vim\vimfiles,  
C:\Program Files\Vim\vim72,  
C:\Program Files\Vim\vimfiles\after,  
C:\Documents and Settings\Jenyay\vimfiles\after
```

То, что можно задавать пути, где будут искаться плагины, очень удобно для упорядочивания установленных дополнений. Например, плагины, прилагающиеся к Vim по умолчанию, лежат в директории *\$VIMRUNTIME*, и туда лучше ничего больше не кидать. Плагины, которые должны работать для всех пользователей, кроме плагинов по умолчанию, лежат в директории *VIM/vimfiles*. А если пользователь не имеет права записи в директорию, где установлен Vim, но ему захочется установить плагин, то он его просто скопирует в *\$HOME/vimfiles*. Плюс есть еще поддиректории *after*, из которых плагины грузятся в последнюю очередь, этот путь могут использовать плагины, которые должны грузиться обязательно после каких-то других плагинов.

Давайте теперь заглянем внутрь этих директорий. В каждой из них есть несколько поддиректорий, но нас в данный момент интересовать будут только некоторые из них, а именно:

- *plugin* - здесь хранятся плагины, которые должны загружаться каждый раз при запуске Vim. Обратите внимание, что плагины запускаются уже после того, как будет прочитан и выполнен файл *.vimrc* / *_vimrc*. Благодаря этому через настройки можно отменять загрузку некоторых плагинов, не удаляя их, но об этом мы еще поговорим.
- *ftplugin* - здесь лежат плагины, которые запускаются только для определенных типов файлов. Благодаря этой возможности разные плагины могут выполнять одинаковые действия для разных типов файлов, или наоборот некоторые плагины будут запускаться только тогда, когда они действительно нужны.

- *autoload* - директория, предназначенная для хранения общих функций, которые могут быть полезны, если потребуется запускать из разных мест. Это, по сути, библиотека скриптов. Основное отличие этой папки от *plugin* заключается в том, что функции в библиотеке можно располагать в более глубоких поддиректориях, а при их вызове нужно будет указывать полный путь, начиная с *autoload*.
- *syntax* - здесь расположены файлы, описывающие раскраску синтаксиса для различных типов файлов.

В данной статье мы не будем рассматривать все типы плагинов, будем делать только общие плагины, которые располагаются в директории *plugin*. Про остальные типы плагинов мы поговорим в следующей части, а может быть даже и не в одной.

А теперь хотелось бы рассмотреть некоторые общие вопросы, которые относятся к большинству типов плагинов.

Оператор `command`

Очень часто плагины добавляют в Vim свои собственные команды, которые могут вызываться из командной строки, поэтому давайте сначала разберемся с тем, как они это делают.

Мы уже сталкивались с созданием команды, когда писали глобальный плагин в [конце пятой части](#) этого цикла статей. Напомню, что тогда мы писали скрипт для автоматического создания оглавления для статей в нотации `rmWiki`.

Для напоминания того, как выглядит создание команды, давайте создадим что-нибудь более простое и короткое, пусть это будет плагин, который просто добавляет в текущий буфер строку с общим количеством строк в буфере.

Назовем файл *calclines.vim* (расширение у всех плагинов независимо от его типа должно быть *.vim*) и поместим его в директорию *plugin*. Вот его текст:

```
function! s:CalcLines ()  
    " Количество строк в буфере  
    let l:count = line('$')  
    call append ('.', l:count)
```

```
endfunction  
command CalcLines call s:CalcLines()
```

[Исходник](#)

Этот очень простой скрипт просто добавляет количество строк на строку, следующую за курсором. Теперь после загрузки этого плагина для его вызова достаточно написать в командной строке Vim:

```
:CalcLines
```

Здесь мы использовали оператор *command* (формально это команда, но получается уж слишком явная тавтология "команда command"), чтобы было проще вызывать функцию *s:CalcLines()*. Также благодаря команде мы функцию сделали локальной внутри скрипта, хотя в данном случае это не большой выигрыш, так как сама команда глобальная. Давайте про этот оператор поговорим подробнее. С виду у него довольно простой синтаксис:

```
:command[!] [{attr}...] {cmd} {rep}
```

Или в сокращенной форме:

```
:com[!] [{attr}...] {cmd} {rep}
```

Здесь *{cmd}* - это имя команды, которую мы хотим создать. Имя команды, которую создает пользователь, обязательно должно начинаться с заглавной буквы.

{rep} - это те действия, которые должна выполнять команда. Здесь могут использоваться специальные параметры, о которых мы еще поговорим. У нас пока в качестве *{rep}* используется простой вызов функции *s:CalcLines()*:

```
call s:CalcLines()
```

Восклицательный знак используется таким же образом, что и при объявлении функции. То есть если существует вероятность того, что такая команда уже существует. В этом случае Vim просто перепишет команду, а без восклицательного знака выдаст ошибку.

Кроме того, после оператора *command* могут быть указаны необязательные атрибуты. Давайте с них и начнем.

Атрибут *-args*

Все атрибуты начинаются с минуса, за которым идет имя атрибута. Один из таких атрибутов *-args* определяет сколько параметров может принимать команда. По умолчанию считается, что команда не принимает никаких параметров (как в нашем случае) или значение атрибута *-args=0*.

-args может принимать следующие значения:

- *-args=0* - команда не принимает параметры.
- *-args=1* - команда может принимать только один параметр.
- *-args=** - команда может принимать любое количество параметров, или может быть запущено вообще без параметров.
- *-args=?* - команда может принимать один параметр или быть запущена без них.
- *-args=+* - команда должна принять как минимум один параметр.

Параметры для команд разделяются пробелами, а не запятыми, как у функций.

Следующая команда с именем *HelloArgs* просто выводит переданные ей параметры.

```
:command! -args=* HelloArgs echo "Arguments:" <args>
```

[Исходник](#)

Теперь в командной строке Vim выполним следующую команду:

```
:HelloArgs "hello" 1 2 4 [10, 20, 30]
```

В результате мы увидим:

```
Arguments: hello 1 2 4 [10, 20, 30]
```

В этом месте вы можете возмутиться, узнав, что параметры разделяются не запятыми, а пробелами, ведь вполне естественно было бы параметры команды передать в какую-нибудь функцию. Для этой цели достаточно воспользоваться вместо `<args>` параметром `<f-args>`, на место которого будет подставлены параметры, разделенные запятой, но взятые в двойные кавычки.

Изменим предыдущий пример, чтобы посмотреть на ту форму, в которой будет создан список параметров:

```
:command! -nargs=* HelloArgs echo "Arguments:" <f-args>
```

[Исходник](#)

Если теперь мы выполним ту же команду

```
:HelloArgs "hello" 1 2 4 [10, 20, 30]
```

то в результате получим:

```
Arguments: "\"hello\"", "1", "2", "4", "[10, ", "20, ", "30]"
```

То, что каждый параметр был обернут еще и в двойные кавычки, может создать дополнительные проблемы, если функция ожидает на входе, например, дробные числа или списки. С целыми числами проблем будет меньше, потому что Vim может автоматически преобразовывать в них строки.

Обратите внимание, что мы не можем задать точное количество обязательных параметров, если их больше 1, что может иногда привести к проблемам, если параметры команды мы захотим передать в функцию, которая ожидает определенное количество параметров, а команде будет передано другое количество параметров. Правда, в этом случае мы все-равно получим сообщение об ошибке, но если бы в этой ошибке явно говорилось о том, что ожидается определенное количество параметров, то жизнь пользователя, допустившего ошибку, была бы немного легче.

Атрибут `-complete`

Другим полезным атрибутом, который можно указать оператору *command*, является атрибут *-complete*, который позволяет использовать автодополнение, срабатывающее при нажатии клавиши Tab, для параметров пользовательской команды.

Для примера пусть мы хотим создать простую команду с именем *FileSize*, которая выводила бы нам размер файла на диске. Первым ее вариантом будет:

```
:command! -nargs=1 FileSize echo getfsize(<f-args>)
```

[Исходник](#)

Теперь, введя следующую команду в командную строку Vim, мы узнаем размер файла в байтах:

```
FileSize c  
    \temp\myfile.txt
```

Но, согласитесь, что каждый раз вводить путь полностью не очень удобно. Добавив всего лишь один аргумент *-complete=file*, после ввода имени команды мы получим автоматическое дополнение имен файлов и папок по нажатию клавиши Tab после имени команды, как в обычной консоли. Теперь наша команда выглядит следующим образом:

```
:command! -nargs=1 -complete=file FileSize echo getfsize(<f-args>)
```

[Исходник](#)

Коротко перечислим некоторые другие значения для аргумента *-complete*:

- *-complete=command* - автодополнение команд командной строки Vim (Ex-режим)
- *-complete=buffer* - автодополнение имен открытых буферов
- *-complete=dir* - автодополнение только имен директорий (*-complete=file* при автодополнении предлагаются как имена файлов, так и директорий)
- *-complete=environment* - автодополнение имен переменных окружения, таких как PATH, HOME, и т.д.
- *-complete=help* - автодополнение заголовков (subject) из справки

- `-complete=tag` - автодополнение тегов Vim

Это не полный список возможных автодополнений, остальные значения аргумента `-complete` вы можете посмотреть в справке (`:h -complete`). Скажу лишь, что не удастся объединить в одной команде автодополнение из разных источников. Если указать несколько атрибутов `-complete`, то использоваться будет последний из перечисленных.

Также есть возможность задавать собственную функцию, которая будет предлагать варианты автодополнения, но обсуждение этого вопроса выходит за рамки этой статьи.

Атрибут `-range` и некоторые другие

Атрибут `-range` мы уже использовали в конце [шестой части](#), поэтому здесь только напомним, что этот атрибут предназначен для того, чтобы созданную команду можно было бы применить к некоторому интервалу строк, как это делается, например, для встроенной команды `s`, когда перед ней ставятся два числа через запятую или символ `%`, обозначающий, что команда должна быть применена ко всем строкам буфера.

С этим атрибутом связаны два параметра: `<line1>` и `<line2>`, которые обозначают, соответственно, начало и конец интервала строк.

Как вы, должно быть уже знаете, некоторые команды Vim могут включать в себя восклицательный знак, который, как правило, обозначает, что команду нужно всегда выполнять, даже если есть некоторые мешающие факторы. Так, например, команда `:q` не будет выполнена, если есть открытый не сохраненный буфер, а команда `:q!` будет выполнена всегда.

Иногда может быть полезным добавить и для своей собственной команды такой знак, чтобы передать его какой-нибудь встроенной команде. Сообщить Vim о том, что команда может принимать восклицательный знак можно с помощью атрибута `-bang`. В этом случае в теле команды можно использовать параметр `<bang>`, который будет заменен на восклицательный знак, если он был у вызываемой команды, или будет удален, если восклицательного знака у команды нет.

Следующий пример, хоть и бесполезный, но показывает использование этого знака. Здесь просто создается команда *MyCommand*, дублирующая команду *:q*.

```
command! -bang MyCommand q<bang>
```

[Исходник](#)

У оператора *command* есть еще некоторые атрибуты, мы их рассматривать не будем, но вы можете прочитать о них в справке (*:h command-bang*)

Соглашения по оформлению плагинов

Пусть вы создали очень полезный плагин, которым вы хотели бы поделиться со всем миром, выложив его, например, на официальный сайт Vim. Для этого вам понадобится предварительно оформить его в принятом стиле (хотя этот принятый стиль довольно свободный). Давайте посмотрим что для этого надо сделать.

Для демонстрации возьмем скрипт, написанный в [шестой части](#). Напомню, что он предназначен для работы со списками дел (ToDoList) и его исходник выглядит следующим образом.

```
function! s:NextStatus() range
  for n in range (a:firstline, a:lastline)
    let line = getline (n)

    if line =~ '\CTODO'
      let line = substitute (line, '\CTODO',
'INPROGRESS', "g")
    elseif line =~ '\CINPROGRESS'
      let line = substitute (line, '\CINPROGRESS',
'DONE', "g")
    elseif line =~ '\CDONE'
      let line = substitute (line, '\CDONE', 'TODO',
'g')
    endif

    call setline (n, line)
  endfor
endfunction

command! -range NextStatus <line1>,<line2>call s:NextStatus()
```

[Исходник](#)

В первую очередь надо убедиться, что все функции скрипта, которые не должны интересовать пользователя, должны быть для него недоступны, и вообще быть доступны только в пределах скрипта. Для таких функций нужно добавить префикс *s:*, что заодно уменьшит вероятность того, что ваша функция начнет конфликтовать из-за имени

с другим плагином. Этому условию наш скрипт удовлетворяет.

Теперь дадим возможность пользователю не загружать наш плагин. Это может быть полезно, если он не является админом на своем компьютере. Сделать это очень просто. Заведем глобальную (!) переменную, которая будет обозначать загружен ли наш плагин (значение переменной при загрузке скрипта станет равным 1) или нет. И в случае, если значение этой переменной будет равно 1, то больше этот плагин не загружать. Так как переменная будет глобальная, то надо аккуратнее подходить к выбору ее имени, чтобы оно более точно описывало наш плагин.

Дополним пример

```
if exists("todo_loaded")
    finish
endif

let todo_loaded = 1

function! s:NextStatus() range
    for n in range(a:firstline, a:lastline)
        let line = getline(n)

        if line =~ '\CTODO'
            let line = substitute(line, '\CTODO',
'INPROGRESS', "g")
        elseif line =~ '\CINPROGRESS'
            let line = substitute(line, '\CINPROGRESS',
'DONE', "g")
        elseif line =~ '\CDONE'
            let line = substitute(line, '\CDONE', 'TODO',
"g")
        endif

        call setline(n, line)
    endfor
endfunction

command! -range NextStatus <line1>,<line2>call s:NextStatus()
```

[Исходник](#)

Что нам это дало? Теперь пользователь может прописать в своем файле настроек .vimrc/_vimrc следующую строку, и наш плагин у него загружаться не будет:

```
let todo_loaded = 1
```

[Исходник](#)

Все условия, которые касается программирования, мы теперь выполнили, теперь осталось документировать наш плагин. Для этого достаточно добавить всего несколько строчек комментариев:

```

" File: todo_next.vim
" Author: Evgeniy Ilin aka Jenyay <jenyay {at} jenyay.net>
" Description: Плагин для последовательного переключения
               статуса задач в TodoList
" Version:     1.1
" Commands: NextStatus

if exists("todo_loaded")
    finish
endif

let todo_loaded = 1

function! s:NextStatus() range
    for n in range(a:firstline, a:lastline)
        let line = getline(n)

        if line =~ '\CTODO'
            let line = substitute(line, '\CTODO',
'INPROGRESS', "g")
        elseif line =~ '\CINPROGRESS'
            let line = substitute(line, '\CINPROGRESS',
'DONE', "g")
        elseif line =~ '\CDONE'
            let line = substitute(line, '\CDONE', 'TODO',
'g')
        endif

        call setline(n, line)
    endfor
endfunction

command! -range NextStatus <line1>,<line2>call s:NextStatus()

```

[Исходник](#)

В принципе, верхний заголовок может содержать любые разделы (первое слово после символа комментария, за которым идет двоеточие, Vim его будет выделять цветом), но очень желательно вписать туда описание плагина в раздел *Description*, автора, версию плагина и дату последней правки. Некоторые авторы туда пишут список изменений в каждой версии программы. В общем, сюда вы можете писать все, что вы считаете важным про ваш плагин.

Вообще-то, желательно эти комментарии писать на английском языке, но я, чтобы не позориться со своим английским, описание написал по-русски.

Документация

Мы уже добавили описание в шапку плагина, оно может быть довольно большим, однако раскраска кода Vim заметно помогает ориентироваться в таком описании, так как выделяет ее разделы цветом. Но все-таки пользователю будет еще удобнее пользоваться вашим плагином, если вы создадите также файл документации. В этом случае он сможет воспользоваться стандартной командой `:help` или `:h`, чтобы вызвать интересующий его раздел про плагин.

Давайте сделаем файл документации для нашего скрипта. Для этого создадим пустой файл с именем *todo_next.txt*. В принципе, имя может быть любым, но лучше, чтобы оно совпадало с именем плагина, а вот расширение должно быть .txt. На самом деле в Vim есть еще такое понятие как локализованная справка, тогда расширение будет .??x, где ?? - это две буквы, обозначающие язык справки, но мы пока будем считать, что мы пишем обычную англоязычную справку (правда, я ее здесь буду писать все-таки по-русски).

Итак, мы создали пустой текстовый файл. Важно, чтобы он был набран в той же кодировке, что и остальные файлы справки, поэтому скорее всего придется использовать какую-нибудь однобайтовую кодировку, иначе потом, при добавлении нашей справки в общую систему помощи, Vim начнет ругаться.

Теперь нужно соблюсти две условности. Во-первых, самая первая строка документации должна начинаться с имени файла документации, окруженного звездочками:

```
*todo_next.txt*
```

[Исходник](#)

Важно, чтобы первая звездочка стояла на первой позиции в первой строке. Вообще говоря, такое выделение слов звездочками в документации делает эти слова разделами, на которые можно ссылаться. Эти разделы очень близки понятию "якорь" в HTML. После того как мы добавим наш файл в общую документацию Vim, введя команду `:h todo_next.txt` мы сможем сразу открыть этот файл документации.

После созданного якоря может идти название плагина, его версия или очень краткое описание, буквально в несколько слов.

В данный момент для Vim это обычный текстовый файл, ничем не примечательный. Однако, если бы он знал о том, что это на самом деле документация, то он смог бы нам раскрасить файл. Для этого достаточно указать Vim, что в данный момент открыт файл справки, сделать это можно с помощью команды

```
:set filetype=help
```

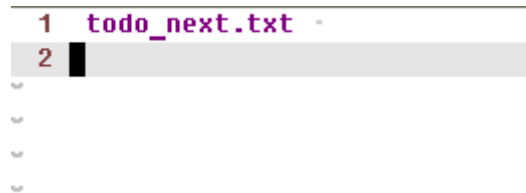
[Исходник](#)

или в сокращенном варианте

```
:set ft=help
```

[Исходник](#)

Теперь, если у нас включена подсветка синтаксиса, то мы увидим, что имя файла *todo_next.txt* выделилось другим цветом, и, возможно, звездочки стали невидимы из-за того, что их цвет стал таким же, как и цвет фона. Например, у меня это выглядит следующим образом:



```
1 todo_next.txt -
2
```

Чтобы нам было еще удобнее, допишем в конец файла новые две строки:

```
*todo_next.txt*
=====
vim:ft=help:tw=78:
```

[Исходник](#)

Строка, состоящая из большого количества знаков "=" - это просто разделитель, в данном случае он будет отделять основное содержание справки от служебной информации (Vim будет выделять цветом и такие разделители, если в нем больше 5 символов "=" или "-"). Служебная информация в данном случае записана в последней строке.

Обозначает она следующее. Эта строка начинается с выражения *vim:*. Теперь Vim при открытии этого файла обратит внимание на это выражение и начнет устанавливать все настройки, указанные далее в этой строке. А в ней мы указали, что для этого файла нужно установить тип как *help*, а потом установили ширину текста как 78 символов (это уж так принято). Дело в том, что для единообразия всей документации желательно, чтобы во всех разделах ширина текста была бы одинаковой. Здесь еще иногда

устанавливают параметр *norl* (*norightleft*), но мы этого делать не будем.

Заметьте, такое поведение Vim дает возможность указывать настройки, которые будут устанавливаться при открытии обычных текстовых файлов. Если не боитесь такого мусора в конце файла, то для них вполне можно устанавливать тип независимо от его расширения. Но не надейтесь таким образом запустить какой-нибудь скрипт или команду, Vim этого сделать не позволит и будет ругаться даже на безвредный *echo*. Таким образом можно только устанавливать настройки.

Теперь заполним основное содержание документации. Пусть это будет следующий текст

```
*todo_next.txt*

1. Description
2. Installation
3. About
4. Version History
=====

DESCRIPTION

Плагин предназначен для быстрой смены состояний.
Для этого используется команда :NextStatus
=====

INSTALLATION

1. Скопируйте файл todo_next.vim в директорию
2. Скопируйте файл todo_next.txt в директорию
3. Выполните команду :helptags ~/.vim/doc (из
директории doc)
=====

ABOUT

Здесь могут быть все контакты автора
=====

VERSION HISTORY

1.1
* Плагин оформлен согласно требованиям к плагиным

1.0
* Первая версия плагина
=====
```

```
vim:ft=help:tw=78:
```

Это будет основное содержание файла справки. Здесь мне пришлось назвать заголовки по-английски, потому сейчас мы начнем оформлять этот текст под синтаксис справки Vim, а он не признает заголовки русскими буквами, и поэтому не подкрашивает их. Но это не страшно, потому что по-хорошему справка должна вся быть на английском языке, если, конечно, вы планируете выкладывать свое творение на официальный сайт Vim.

Как вы, наверное, заметили, в документации есть оглавление, которое должно ссылаться на разделы. Выше уже говорилось, что выражение между звездочками воспринимается Vim аналогично якорю в HTML. Давайте добавим эти якоря к каждому из разделов. Они должны располагаться на той же строке, что и заголовки.

```
*todo_next.txt*
```

```
1. Description
2. Installation
3. About
4. Version History
```

```
=====
```

DESCRIPTION

```
Плагин предназначен для быстрой смены состояний.
Для этого используется команда :NextStatus
```

```
=====
```

INSTALLATION

```
1. Скопируйте файл todo_next.vim в директорию
2. Скопируйте файл todo_next.txt в директорию
3. Выполните команду :helptags ~/.vim/doc (или
   директорию doc)
```

```
=====
```

ABOUT

```
Здесь могут быть все контакты автора
```

```
=====
```

VERSION HISTORY

```
1.1
```

```
* Плагин оформлен согласно требованиям к плаги
```


Якоря мы добавили, теперь можно добавлять ссылки на них. Ссылки оформляются аналогично якорям с той лишь разницей, что вместо звездочек используются символы вертикальной черты.

```
*todo_next.txt*

1. Description |todo-description
2. Installation |todo-installation
3. About |todo-about
4. Version History |todo-version-history
=====

DESCRIPTION

Плагин предназначен для быстрой смены состояния
Для этого используется команда :NextStatus
=====

INSTALLATION

1. Скопируйте файл todo_next.vim в директорию
2. Скопируйте файл todo_next.txt в директорию
3. Выполните команду :helptags ~/.vim/doc (или
   директорию doc)
=====

ABOUT

Здесь могут быть все контакты автора
=====

VERSION HISTORY

1.1
* Плагин оформлен согласно требованиям к плагинам

1.0
* Первая версия плагина
=====

vim:ft=help:tw=78:
```

Ссылки тоже будут подкрашиваться

Теперь в качестве небольшого украшения выделим номера версий в последнем разделе как подразделы (без ссылок на них). Для этого достаточно добавить после раздела знак тильды:

1.1~
* Плагин оформлен согласно требованиям к плаги

1.0~
* Первая версия плагина

19/22

[illegible]

Мы еще не добавили наш файл справки в общую документацию, поэтому ссылки на якоря пока еще не работают. Давайте исправим это недоразумение.

Скопируем наш файл справки в директорию *doc*, пусть это будет для примера *\$VIM/vimfiles/doc*, теперь запустим Vim. После этого нужно выполнить команду

```
:helptags $VIM/vimfiles/doc
```

Эта команда скажет Vim, что нужно заново пересчитать все файлы документации и пересобрать файл тегов. Обычно эта команда выполняется быстро, хотя может занять секунду-другую.

Собственно, все. Теперь мы можем выполнить следующую команду, чтобы увидеть справку по программе.

```
:help todo next.txt
```

Кроме того, если вы скопировали документацию в ту же папку *vimfiles/doc*, то ссылка на наш файл документации появилась в разделе *local-additions*. Убедиться в этом вы можете, набрав следующую команду:

```
:help local-additions
```

```
LOCAL ADDITIONS:                                local-additions
perlsupport.txt                                Perl Support                                March 16 2009
snippets_emu.txt                                For Vim version 7.0. Last change: 2006 Dec 26
tComment.txt                                tComment -- An easily extensible & universal comment plugin
todo_next.txt                                Vim Outliner OF Folds
voof.txt
```

Если бы в первой строке мы бы еще написали какую-нибудь фразу, то она бы тоже была бы выведена здесь рядом со ссылкой на документацию.

На этом пока прервемся, скачать архив с плагином `todo_next` и файлом документации к нему вы можете [здесь](#).

В следующей части (а может быть частях) мы рассмотрим как создавать плагины, которые будут загружаться только для файлов определенного типа. Также мы еще не рассмотрели то как можно создавать библиотеки скриптов и плагины для раскраски синтаксиса.

[Часть 9. Другие типы плагинов](#)

Вы можете подписаться на новости сайта через [RSS](#), [Группу Вконтакте](#) или [Канал в Telegram](#).

★★★★★

Рейтинг 4.9/5. Всего 16 голос(а, ов)

☐ Плохо ☐ Так себе ☐ Неплохо ☐ Хорошо ☐ Отлично

Голосовать

TheAthlete 23.04.2012 - 17:05

Документация

Здравствуй! Не подскажите, как в справку добавить ссылку на другой файл, в том числе, который идет в стандартной поставке.

Например, когда я пишу

```
|'runtimepath'|
```

должна открываться справка по `runtimepath` при вызове `Ctrl-]`



[Подписаться на комментарии](#)

Автор:

Тема:

Ваш комментарий

I

B

U

[^]

_˘

x^2

x_2

h

Ab

[@

☺

☺

☹

???

🙄

😬

😏

😈

😄

Введите код

423

Послать

© Евгений Ильин 2008-2024 (jenyay.ilin@gmail.com)

<https://jenyay.net/Programming/VimScript8>

22/22