РУКОВОДСТВО ПО ЯЗЫКУ СКРИПТОВ

Это руководство является введением в мощный встроенный язык скриптов для RouterOS.

В качестве полезных данных следует привести ссылки на страницы из официальной документации:

1. Синтаксис языка — оригинал данной статьи
2. Полезные трюки при написании скриптов;
3. Набор готовых скриптов по категориям.

Скриптовый язык routeros предоставляет возможность для автоматизации задач средствами исполнения определённых пользователем скриптов, связанных с какими-либо событиями.

Скрипты могут могут храниться в репозитории (хранилище) или могут быть написаны прямо в консоли. События, вызывающие исполнение скриптов генерируются системным планировщиком, утилитой мониторинга траффика и утилитой netwatch, но не ограничены только этими генераторами.

СТРУКТУРА КОМАНДЫ

Скрипты для RouterOS состоят из команд. Команды исполняются одна за одной, пока не будет достигнут конец скрипта или не возникнет ошибка во время исполнения.

Команда

Консоль использует следующий синтакс команды:

```
[prefix] [path] command [uparam] [param=[value] .. param=[value]]
```

[prefix] — может принимать следующие символьные значения, «:» или «/», что определяет команду, как ICE или путь.
[path] — относительный путь к желаемому уровню меню.
command — одна из возможных встроенных команд, определённых уровнем меню.
[uparam] — безымянный параметр, кот. должен быть определён, если этого требует встроенная команда (command).
[params] — последовательность именованных параметров со значениями.

* Всё, что перечислено в квадратных скобках является необязательными составными частями команды.

Каждая команда заканчивается символом «;» или концом строки. В некоторых случаях эти символы не нужны для окончания команды. Команда внутри (), [] или {} не нуждается в таких символах.

```
:if ( true ) do={ :put "lala" }
```

Каждая команда внутри другой команды начинается и заканчивается квадратными скобками [] (объединение команд):

```
:put [/ip route get [find gateway=1.1.1.1]];
```

Команда может состоять из нескольких строк, объединённых специальным символом. Смотри «Объединение команд».

Команда и EOL (End of Line)

Команда — это последовательность символов, заканчивающихся EOL последовательностью. Любая из стандартных последовательностей EOL может быть использована:

— Unix = ASCII LF
— Windows = ASCII CR LF
— Mac = ASCII CR

Комментарии

Комментарий начинается с символа «#» и заканчивается последовательностью EOL. Пробел и любые другие символы не разрешены к использованию перед #. Комментарии игнорируются синтаксическим анализатором. Если символ # появится в строке, то это не будет считаться комментарием:

```
# хороший комментарий
# плохой комментарий
:global a; # плохой комментарий
:global myStr "лала # это не комментарий"
```

Объединение команд

Две или более строки могут быть объединены в одну команду, если использовать символ «\» (обратный слэш). Это не работает с комментариями и с токенами исключая строки. Примеры:

```
:if ($a = true \
and $b=false) do={ :put "$a $b"; }
:if ($a = true \ # неудачная попытка
and $b=false) do={ :put "$a $b"; }
# comment \
```
continued – invalid (синтаксическая ошибка)

Пробелы между токенами

Пробелы используются для разделения токенов. Пробел необходим между двумя токенами только если их объединение может быть интерпретировано, как другой токен. Пример:

```
{
:local a true; :local b false;
# пробелы не нужны
:put (a&&b);
# пробелы нужны
:put (a and b);
}
```

Пробелы не разрешены в следующих конструкциях:

— =
— from=, to=, step=, in=, do=, else=

Пример:

```
#неправильно:
:for i from = 1 to = 2 do = { :put $i }
#правильно:
:for i from=1 to=2 do={ :put $i }
:for i from= 1 to= 2 do={ :put $i }
```

```
#неправильно
/ip route add gateway = 3.3.3.3
#правильно
/ip route add gateway=3.3.3.3
```

Области видимости

Переменные могут быть использованы только в определённых областях скрипта. Эти области называют областями видимости. Существует два типа областей видимости: локальная и глобальная. Переменная, объявленная внутри блока может быть использована только в его пределах после места объявления.

Глобальная область видимости, по другому корневая область, видимости является областью по умолчанию для скрипта. Она создаётся автоматически и существует всегда.

Пользователь может определить свои области видимости при помощи скобок {}. Такие области называются локальными. Пример:

```
{
:local a 3;
{
:local b 4;
:put ($a+$b);
}
# команда ниже некорректна, так как переменная b не определена в данной
области видимости
:put ($a+$b);
}
```

Важное замечание: каждая команда в консоли обрабатывается в своей локальной области видимости.

```
[admin@MikroTik] > :local myVar a;
[admin@MikroTik] > :put $myVar
syntax error (line 1 column 7)
```

Важно: не определяйте глобальные переменные внутри локальных областей видимости.

```
# этот код сгенерирует ошибку
{
:local a 3;
{
:global b 4;
}
:put ($a+$b);
}
```

ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА

Следующие слова являются зарезервированными и не могут быть использованы, как имена переменных или функций:

```
and or not in
```

ОГРАНИЧИТЕЛИ

Следующие токены служат ограничителями в грамматике:

```
() [] {} : ; $ /
```

ТИПЫ ДАННЫХ

Язык имеет следующие типы данных:

— num — 64 битное знаковое целое. может быть инициализировано шестнадцатиричным значением
— bool — булев тип. может быть true или false
— str — строковый тип. последовательность символов
— ip — IP адрес
— ip6-prefix — IPv6 префикс
— id — шестнадцатиричное значение с префиксом *. Каждая позиция меню имеет уникальный внутрений ID
— time — дата и время
— array — последовательность значений, массив
— nil — тип переменной по умолчанию, если она не инициализирована никаким значением

Escape последовательности

Следующие последовательности могут быть использованы для помещения спецсимволов в строки:

\» — двойные кавычки
\\ — обратный слэш
\n — символ конца строки
\r — return
\t — горизонтальная табуляция
\$ — знак доллара
\? — знак вопроса. спецназначение этого символа — выводить справку меню
_ — пробел
\a — звуковой сигнал терминала BEL (0x07)
\b — бэкспейс
\f — form feed (0xFF)
\v — вертикальная табуляция
\xx — шестнадцатиричный код символа

Пример:

```
:put "\48\45\4C\4C\4F\r\nThis\r\nis\r\na\r\ntest";
```

Вывод:
```
HELLO
This
is
a
test
```

ОПЕРАТОРЫ

Арифметические операторы

+ бинарное сложение
— бинарное вычитание
* бинарное умножение
/ бинарное деление
— унарное отрицание

Примеры:

```
:put (3+4);
:put (3-4);
:put (3*4);
:put (10 / 2); :put ((10)/2)
{ :local a 1; :put (-a); }
```

Операторы сравнения

< меньше
> больше
= равно
<= меньше или равно
>= больше или равно
!= не равно

Логические операторы

`(!, not)` НЕ
`(&&, and)` И
`(||, or)` ИЛИ
`(in)` оператор принадлежности (пример, :put (1.1.1.1/32 in 1.0.0.0/8);)

Битовые операторы

Применимы к числовому типу и типу IP address.

| ИЛИ
& И
~ Инверсия
^ XOR
<< сдвиг влево
>> сдвиг вправо

Примеры. Вычисление адреса подсети по IP адресу и маске

```
{
:local IP 192.168.88.77;
:local CIDRnetmask 255.255.255.0;
:put ($IP&$CIDRnetmask);
}
```

Получение последних 8 бит IP адреса

```
:put (192.168.88.77&0.0.0.255);
```

Вычисление широковещательного адреса

```
{
:local IP 192.168.88.77;
:local Network 192.168.88.0;
:local CIDRnetmask 255.255.255.0;
:local InvertedCIDR (~$CIDRnetmask);
:put ($Network|$InvertedCIDR)
}
```

Операторы конкатенации

конкатенация двух строк `:put (“concatenate” . “ “ . “string”);`
конкатенация двух массивов или объявление значения в массив `:put ({1;2;3} , 5 );`

Также возможно добавлять значения переменных в строки без конкатенации

```
:global myVar "world";
:put ("Hello " . $myVar);
# следующая команда делает тоже, что предыдущая
:put "Hello $myVar";
```

Используя $[] и $() можно добавлять выражения внутрь строк

```
:local a 5;
:local b 6;
:put "5x6 = $($a * $b)";
:put "We have $[ :len [/ip route find] ] routes";
```

Другие операторы

`[]` Использование команды в команде
`()` Группировка выражений
`$` Унарный оператор подстановки. доступ к значению переменной
`~` Бинарный оператор, сравнивающий значение с POSIX регулярным выражением
`->` Получение элемента массива по ключу

Пример с регулярными выражениями

```
# печатаем все маршруты со шлюзом, у которого адрес заканчивается на 202
/ip route print where gateway~"^[0-9\\.]*202"
```

Пример с массивом

```
[admin@x86] >:global aaa {a=1;b=2}
[admin@x86] > :put ($aaa->"a")
1
[admin@x86] > :put ($aaa->"b")
2
```

ПЕРЕМЕННЫЕ

Язык допускает два типа переменных, глобальные и локальные.

global — доступно из всех скриптов, созданных текущим пользователем.
local — доступно только внутри текущей области видимости.

Замечание: значение переменной ограничено 4096 байтами.

Каждая переменная, исключая встроенные, должна быть объявлена перед использованием при помощи зарезервированных слов global или local.

```
# некорректный код
:set myVar "my value";
:put $myVar

# корректный код
:local myVar;
```

```
:set myVar "my value";
:put $myVar;
```

Исключая случай, когда используется множество (set) переменных. Пример

```
/system script
add name=myLeaseScript policy=\
ftp,reboot,read,write,policy,test,winbox,password,sniff,sensitive,api \
source=":log info \$leaseActIP\r\
\n:log info \$leaseActMAC\r\
\n:log info \$leaseServerName\r\
\n:log info \$leaseBound"
/ip dhcp-server set myServer lease-script=myLeaseScript
```

Допустимыми символами для составления имён переменных являются буквы и цифры. Если имя содержит любые другие символы, то его нужно помещать в двойные кавычки «».

```
# корректно
:local myVar;
# некорректно
:local my-var;
# корректно
:global "my-var";
```

Если переменная объявлена, но не инициализирована, то ей назначается тип nil. Иначе тип определяется автоматически. Иногда необходимо преобразовать переменную из одного типа в другой. Это можно сделать при помощи специальных команд.

```
# преобразование строки в массив
:local myStr "1,2,3,4,5";
:put [:typeof $myStr];
:local myArr [:toarray $myStr];
:put [:typeof $myArr]
```

Имена переменных чувствительны к регистру. Команда set без значения удаляет переменную из окружения.

```
:global myVar "myValue"
:set myVar;
```

ВСТРОЕННЫЕ КОМАНДЫ

Навигация и справка

1) `/` — переход в корень меню

2) `..` — переход в меню на один уровень выше

3) `?` — список всех возможных команд меню с описанием

Команды общего назначения

Любая встроенная команда начинается с символа «:», иначе она будет воспринята, как переменная.

1) global — определение глобальной переменной

Синтаксис:
`:global []`

Пример:
```
:global myVar "something"; :put $myVar;
```

2) local — определение локальной переменной

Синтаксис:
`:local []`

Пример:
```
{ :local myLocalVar "I am local"; :put $myVar; }
```

3) beep — звуковой сингнал из встроенного спикера

Синтаксис:
`:beep`

Пример:
```
:beep frequency=320 length=100ms;
```

4) delay — задержка заданный период времени

Синтаксис:
`:delay`

Пример:
```
:delay 50ms;
```

5) put — вывод в консоль

Синтаксис:
`:put`

6) len — печать длины переданой последовательности (массива или строки)

Синтаксис:
`:len`

Пример:
```
:put [:len "length=8"];
```

7) typeof — определение типа переменной

Синтаксис:
`:typeof`

Пример:
```
:put [:typeof 4];
```

8) pick — вернуть подпоследовательность массива или строки. если неопределено, то вернёт элемент из позиции

Синтаксис:
`:pick []`

Пример:
```
:put [:pick "abcde" 1 3]
```

9) log — запись в системный лог. topic={debug, error, info, warning}

Синтаксис:
```
:log
```

Пример:
```
:log info "Hello from script";
```

10) time — определить время, необходимое для выполнения команды

Синтаксис:
```
:time
```

Пример:
```
:put [:time {:for i from=1 to=10 do={ :delay 100ms }}];
```

11) set — присвоить значение переменной

Пример:
```
:global a; :set a true;
```

12) find — вернуть позицию искомого в строке или массиве

Синтаксис:
```
:find
```

Пример:
```
:put [:find "abc" "a" -1];
```

13) environment — печать информации об инициализированных переменных

Синтаксис:
```
:environment print
```

Пример:
```
:global myVar true; :environment print;
```

14) terminal — ?

15) error — сгенерировать ошибку в консоль и остановить выполнение

Синтаксис:
```
:error
```

16) execute — выполнение скрипта в фоне

Синтаксис:
```
:execute
```

Пример:
```
    :local j [:execute {/interface print follow where [:log info ~Sname~]}];
        :delay 10s;
        :do { /system script job remove Sj } on-error={}
```

17) parse — парсинг текста в набор команд для исполнения. можно использовать для определения функций

Синтаксис:
```
:parse <text>
```

Пример:
```
:global myFunc [:parse ":put hello!"];
    $myFunc;
```

18) resolve — разрешение имени в IP адрес

Пример:
```
:put [:resolve "www.mikrotik.com"];
```

* Команды преобразования типов *

1) toarray <var>
2) tobool <var>
3) toid <var>
4) toip <var>
5) toip6 <var>
6) tonum <var>
7) tostr <var>
8) totime <var>

Команды, специфичные для меню

Следующие команды доступны в большинстве подменю и не требуют использования символа «:»

1) add — добавить запись
   Синтаксис:
   ```
   add <param>=<value>..<param>=<value>
   ```

2) remove — удалить запись
   Синтаксис:
   ```
   remove <id>
   ```

3) enable — активировать запись
   Синтаксис:
   ```
   enable <id>
   ```

4) disable — дизактивировать запись
   Синтаксис:
   ```
   disable <id>
   ```

5) set — изменение параметров записи с заданным ID
   Синтаксис:
   ```
   set <id> <param>=<value>..<param>=<value>
   ```

6) get — ?
   Синтаксис:
   ```
   get <id> <param>=<value>
   ```

   Пример:
   ```
   /system script get myScript source
   ```

7) print — печать записей. вывод зависит от параметров

Синтаксис:
`print <param><param>=[<value>]`

8) export — сохранить записи в файл в виде скрипта, если файл указан. иначе вывести записи в окно терминала
Синтаксис:
`export [file=<value>]`

9) edit — редактирование выделеной записи во встроеном текстовом редакторе
Синтаксис:
`edit <id> <param>`

10) find — вернуть список ID записей, которые совпадают с expression.
Синтаксис:
`find <expression>`

Пример:
`:put [/interface find name~"ether"]`

Возможный вывод:
*1;*2;*3;*4

Импорт

Команда import доступна из корневого меню и используется для импорта конфигураций из файлов, созданных командой export или вручную.

Параметры команды print

append —
as-value — вернуть, как массив параметров и их значений (:put [/ip address print as-value])
brief — печать в сокращённом табличном виде
advanced —
detail — детальная печать, отображает все параметры
count-only — печатать количество записей, а не сами записи
file — перенаправить печать в файл
follow — печать текущих записей и слежение за появлением новых записей, пока не будет нажата комбинация ctrl+c (/log print follow)
follow-only — печать и слежение только за новыми записями. выход также по ctrl+c
from — печатать параметры только указанной записи (/user print from=admin)
interval — непрерывно печатать через определённые интервалы времени
terse — отобразить в компактном, удобном для программной обработки виде
value-list — отобразить параметры и значения по одному на линии. удобно для парсинга
without-paging — печать информации без разбиения на части
where — фильтр по значению какого-либо параметра

Одновременно можно использовать несколько параметров. Пример:

```
# печатать текущее количество маршрутов через интерфейс ether1 каждую секунду
/ip route print count-only interval=1 where interface="ether1"
```

ЦИКЛЫ И ВЕТВЛЕНИЯ

Циклы

— while и do-while

```
:while ( <conditions> ) do={ <commands> };
:do { <commands> } while=( <conditions> );
```

— for

```
:for <var> from=<int> to=<int> step=<int> do={ <commands> }
```

— foreach

```
:foreach <var> in=<array> do={ <commands> };
```

* Ветвление *

```
:if(<condition>) do={<commands>} else={<commands>}
```

Пример:

```
{
:local myBool true;
:if ($myBool = false) do={ :put "value is false" } else={ :put "value is true" }
}
```

ФУНКЦИИ

Язык не позволяет создавать функции напрямую, однако можно использовать команду :parse, как обходной способ для создания функций. Начиная с версии 6.2 добавлен новый синтаксис, позволяющий определять функции и передавать им параметры. Также возможно возвращать значение из функции при помощи команды :return. Пример:

```
# создание и выполнение простой функции
:global myFunc do={:put "hello from function"}
$myFunc
```

Вывод:

```
hello from function
```

```
# передача аргументов в функцию
:global myFunc do={:put "arg a=$a"; :put "arg '1'=$1"}
$myFunc a="this is arg a value"  "this is arg1 value"
```

Вывод:

```
arg a=this is arg a value
arg '1'=this is arg1 value
```

Из примера видно, что существуют два способа передачи аргументов в функцию:

— передача по имени
— передача по порядковому номеру

Пример использования :return

```
:global myFunc do={ :return ($a + $b)}
:put [$myFunc a=6 b=2]
```

Вывод:

```
8
```

Также возможно копирование существующего скрипта из окружения и использование его, как функции. Пример:

```
#add script
   /system script add name=myScript source=":put \"Hello $myVar !\""

:global myFunc [:parse [/system script get myScript source]]
   $myFunc myVar=world
```

Вывод:

```
Hello world !
```

Внимание: если функция содержит глобально определённую переменную с именем совпадающим с именем передаваемого параметра, то глобально определённая переменная будет проигнорирована, для совместимости со старыми версиями. Эта возможность может быть изменена в будущих версиях. Избегайте использование параметров с теми же именами, что и глобальные переменные. Пример:

```
:global my2 "123"
   :global myFunc do={ :global my2; :put $my2; :set my2 "lala"; :put
$my2 }
   $myFunc my2=1234
   :put "global value $my2«
```

Вывод:

```
1234
   lala
   global value 123
```

Чтобы вызвать функцию внутри другой функции, её имя должно быть определено.

```
:global funcA do={ :return 5 }
   :global funcB do={
     :global funcA;
     :return ([$funcA] + 4)
   }
   :put [$funcB]
```

Вывод:

```
9
```

ПЕРЕХВАТ И ОБРАБОТКА ОШИБОК ВО ВРЕМЯ ИСПОЛНЕНИЯ

Начиная с версии 6.2 скрипты могут перехватывать ошибки, возникающие во время выполнения. Рассмотрим следующий скрипт:

```
{ :put [:resolve www.a.com]; :put "lala"; }
   failure: dns name does not exist
```

Мы хотим перехватить эту ошибку и обработать её:

```
:do {
          :put [:resolve www.a.com]
   } on-error={ :put "resolver failed"};
   :put "lala"
```

Вывод:

```
resolver failed
   lala
```

ОПЕРАЦИИ С МАССИВАМИ

Внимание: Если имя ключа содержит символы, отличающиеся от латинских в нижнем регистре, то такое имя нужно заключать в двойные кавычки. Пример:

```
{:local a { "aX"=1 ; ay=2 }; :put ($a->"aX")}
```

Конструкция foreach может быть использована для циклического прохода элементам массива.

```
 :foreach k,v in={2; "aX"=1 ; y=2; 5} do={:put ("$k=$v")}
```

Вывод:

```
0=2
1=5
aX=1
y=2
```

Если элементы массива имеют ключевую часть, то они сортируются в алфавитном порядке (alphabetical order) по ключу. Элементы без ключевой части располагаются до элементов с ключевой частью и их порядок остаётся неизменным.

Если конструкиця foreach используется с одним аргументом, то возвращаются значения элементов.

```
:foreach k in={2; "aX"=1 ; y=2; 5} do={:put ("$k")}
```

Вывод:

```
2
   5
   1
   2
```

Пример изменеия значения элемента массива:

```
:global a {x=1; y=2}
   :set ($a->"x") 5
   :environment print
```

Вывод:

```
a={x=5; y=2}
```

РЕПОЗИТОРИЙ (ХРАНИЛИЩЕ) СКРИПТОВ

Уровень меню: /system script

Содержит скрипты, созданные пользователями. Скрипты могут быть выполнены в следующих случаях:

— по событию
— один скрипт вызывает другой скрипт
— запуск вручную

Параметры скриптов:

— comment (тип: string, значение по умолчанию: «») — комментарий, облегчающий понимание, что делает скрипт
— name (тип: string, значение по умолчанию: «Script[num]») — уникальное имя
— policy (тип: string, значение по умолчанию: «») — список применяемых политик

* api — разрешает использование API
* ftp — разрешает удалённый доступ по ftp, передачу файлов на и с роутера
* local — разрешает локальный вход через консоль
* password — разрешает смену паролей
* policy — управление пользователями, добавление/удаление
* read — разрешение чтения конфигурации
* reboot — разрешение на перезагрузку
* sensitive — просмотр паролей и прочей информации
* sniff — разрешение использовать sniffer, torch и тд
* ssh — разрешение на удалённый логин по ssh
* telnet — разрешение на удалённый логин по telnet
* test — разрешение использовать ping, traceroute, bandwidth test
* web — разрешение на удалённый логин по http
* winbox — разрешает использование winbox
* write — разрешение изменения конфигурации

— source (тип: string, значение по умолчанию: «») — код скрипта

Параметры состояния, доступные только для чтения:

— last-started (тип: date) — дата и время последнего запуска
— owner (тип: string) — владелец скрипта
— run-count (тип: integer) — количство запусков

Команда запуска:

```
run [id | name]
```

Окружение

Уровень меню: /system script environment или /environment

Содержит все переменные, определённые пользователем и их значения. Пример:

```
[admin@MikroTik] > :global example;
   [admin@MikroTik] > :set example 123
   [admin@MikroTik] > /environment print
```

Вывод:

```
"example"=123
```

Параметры только для чтения:

* name (тип: string)
* user (тип: string)
* value

Список выполняющихся скриптов

Уровень меню: /system script job

Содержит список всех скриптов, выполняющихся в данный момент.

Параметры только для чтения:

* owner (тип: string)
* policy (тип: array)
* started (тип: date)