Scripting Examples

Document revision 1 (Wed Dec 22 13:15:36 GMT 2004)
This document applies to V2.8

Table of Contents

General Information

Summary

This section contains various scripting examples.

Related Documents

- VRRP
- DHCP Client and Server
- HotSpot Gateway
- Universal Client Interface
- AAA
- IP Pools
- Scripting Host and Complementary Tools

Application Examples

Creating many firewall rules

To make 100 entries in firewall **input** rule making it to accept all packets from addresses starting with 1.1.1.1 till 1.1.1.100:

```
:for e from 1 to 100 do={
  /ip firewall rule input add \
    src-address=(1.1.1. . $e)\
    src-netmask=255.255.255.255
}
```

Getting bandwidth test values

This example shows how to get **bandwidth-test** command's results. As the example uses global variables, other scripts run at the same time may get the current TX values reported by the command.

```
/system script add name=bandtest source={
  :global i
  /tool bandwidth-test 1.1.1.1 \
    direction=transmit \
    duration=14s \
    do={
      :if ($status="running") do={
        :set i $tx-current
        }
      }
  }
```

Using comments

It is sometimes more convinient to refer to an entry by its name. But what to do, if name is not available for an entry (for example, in firewall or in routing)? The example shows how to use comments, which are available for any router's entry table.

Suppose, we need to switch routing tables depending on the reachability of the **10.0.0.217** host. First, we need to print the policy routing rule table, and choose the entry number you need to change. We will use number 0 in this example:

```
[admin@MikroTik] ip policy-routing rule> set 0 comment=com
```

Now, the actual configuration:

```
[admin@MikroTik] system script> add name=gw_1 source={
  /ip policy-routing rule set \
    [/ip policy-rout find comment=com] \
    table=gw1
  }
[admin@MikroTik] system script> add name=gw_2 source={
  /ip policy-routing rule set \
    [/ip policy-rout find comment=com ] \
    table=gw2
  }
[admin@MikroTik] system script> /tool netwatch
[admin@MikroTik] tool netwatch> add host=10.0.0.217 interval=10s timeout=998ms \
\... up-script=gw_1 down-script=gw_2
```

In this case we're pinging host 10.0.0.217 every 10s and if no response is received, the routing table is changed.

Sending files from router

To send a backup file from a router every 7 days to the **example@example.com** e-mail address:

```
/system script add name=e-backup source={
  /system backup save name=email
  /tool e-mail send \
    to="example@example.com" \
    subject=([/system identity get name] . " Backup") \
    file=email.backup
  }
/system scheduler add interval=7d name="email-backup" on-event=e-backup
```

Execute script until reboot

Sometimes it is needed to execute script until the router is rebooted. To do this you may use an endless loop. For example:

```
/system script add name=loop source={
  :while true do={
    :log message="example"
    :delay 10s
    }
  }
```

Here we use 10 second long delays after executing the needed command (in this example, we just write a **example** example message to the system log).

Run the script as usual - with a **/system script run loop** command. You may stop in from the **/system script job** submenu (with the **remove** command).

A ping-based netwatch

This is how you can make something like NetWatch by yourself. For example, we will send e-mail if five pings will fail:

```
/system script add name=pinging source={
  :if ([/ping 1.1.1.1 count=5] = 0) do={
    /tool e-mail send \
      to=example@example.com \
      subject="Can't ping 1.1.1.1"
    }
  }
```

Application Examples

Monitoring DHCP client's address

In this example, we will monitor IP address on the **ether1** interface, and if it is changed, send an e-mail:

```
/system script add name="changed-address" source={
  :if ([/system scheduler get check run-count]<=1) do={
    /system script run start-once
    }
  :global temp
  :global b
  :set temp $a
  :set b [ \
    /ip address get \
      [/ip address find=interface=ether1] \
      address \
    ]
  :if ($temp != $b) do={
    /tool e-mail send \
      to=example@example.com \
      subject="The dynamic IP gets changed"
    :set a $b
    }
  }

/system script add name="start-once" source={
  :global a
  :set a [ \
    /ip address get \
      [/ip address find interface=ether1] \
      address \
    ]
  }

/system scheduler add name=check interval=1m on-event=changed-address
```

Monitoring interface status

The example writes a message in system's log if the **ether1** interface is not disconnected (e.g., when cable is not working, or plugged out). You may use it to constantly monitor interfaces' status.

```
:global u
/interface ethernet monitor ether1 once do={
  :set u $status
  }
:if ($u != "link-ok") do={
  :log message="interface ether1 is disconnected"
  }
```

Other similar example is to do a reboot once cable is disconnected from the router:

```
:global u
/interface ethernet monitor ether1 once do={
  :set u $status
  }
:if ($u != "link-ok") do={
  /system reboot
  }
```

To save interface statistics to a file you will need to perform the following tasks:

- Make a system script that will write to the logs interface status.
  `[admin@MikroTik] system script> add name=eth_stat_to_log source={ /interface`

```
ethernet monitor ether1 once do={:log \ \... message=("\[My log 001\] :: Eth1
status: " . $status)}}
```

- Attach this script to a system scheduler event.
```
[admin@MikroTik] system scheduler> add interval=1h name="int_to_log" \ \...
on-event=eth_stat_to_log
```

- Configure **System-Info** facility to log all information to disk.
```
/system logging facility set System-Info=disk
```

Attention! Enabled local disk logging leads to regular writes to the local file storage. This may result in a faster wearout of solid state medium (CF and IDE-Flash cards).

Watching network performance

Run this script to make a system LED be lit proportionally to the data rate received by the ether1 interface (i.e., the LED is lit for 10 * data_rate_in_megabits_per_second miliseconds):

```
/ system script add name="led" source={
  :local rx
  /interface monitor-traffic ether1 interval=1s do={
    :set rx ($received-bits-per-second/1048576)
    :if ($rx!=0) do={
      :led led2=yes length=((10*$rx) . "ms")
      } \
    else={
      :log message="There is no traffic received on this interface"
      }
    }
  }
```

Notify on data rate records

Running this script in scheduler every seconds, you will get reports on data rate highest records. This may be useful for monitoring peak speeds through interfaces.

```
/system script add name="record" source={
  :global tmp
  :global tx
  :global rx
  :foreach i in [/interface find] do={
    /interface monitor-traffic $i once do={
      :set tx ($sent-bits-per-second/1048576)
      :set rx ($received-bits-per-second/1048576)
      :if ([/system scheduler get record run-count]=1) do={
        :global ttx
        :set ttx $tx
        :global trx
        :set trx $rx
        }
      :if ($tx>$ttx) do={
        /tool e-mail send \
          to=example@example.com \
          subject="Script message" \
          body=("The transmission traffic on " . \
          [/interface get $i name] . " got up to " . $tx . "Mbps")
        :set ttx $tx
        }
      :if ($rx>$trx) do={
        /tool e-mail send \
          to=example@example.com \
          subject="Script message" \
          body=("The receiving traffic on " . \
          [/interface get $i name] . " got up to " . $rx . "Mbps")
```

```
        :set trx $rx
        }
      }
    }
  }
```

Notify on AP change

This script if run from scheduler will notify you in case AP's MAC address change for any of your wlan cards switched in **station** mode.

```
/ system script add name="mail_on_AP_change" source={
  :global mac
  :foreach i in [/interface wireless find mode=station] do={
    /interface wireless monitor $i once do={
      :set mac $bssid
      :if ([/system scheduler get mail_on_AP_change run-count]=1) do={
        :global tmp
        :set tmp $mac
        :if ($mac!=$tmp) do={
          /tool e-mail send \
            to=example@example.com \
            subject=("AP has changed for the " . \
            [/interface wireless get $i name] . " interface") \
            body=("Client AP's MAC address has changed to " . $mac
        }
      }
    }
  }
```

Application Examples

Description

First of all, note that IP protocol does not allow much tricks unless you control both sides of a link. Every such a trick is limited by routing - you must make packets be able to run out of your network, as well as to return back correctly, so that both peers can talk to each other. In most cases, the particular path is determined by the IP address you use, so you should take care that the packets sent out to the Internet will be able to return from the same channel. The only exception is the use of dynamic routing protocols, such as BGP or OSPF (which are not covered by this document) - these protocols may provide load-balancing and fail-over by themselves.

The **fail-over** terms may mean two things:

1. in case the default gateway of your router does not respond (i.e., the channel has been broken, or the gateway is down), you will switch all the traffic to another gateway
2. in case your router has crashed, a backup router will take care of your network's traffic (this is also known as high availability, which is provided by VRRP protocol)

More complicated term is **load balancing**. We can differentiate between at least four kinds of traffic-treating that may be called "load-balancing":

1. NAT with per session balancing (means no servers on the router side will get any benefit of balancing unless they use DNS-based load-balancing) - each session is tracked and routed through the same channel until the connection is closed. Each new connection is sent through the next route (you can

have more than two). This is supported by MikroTik RouterOS.

2. balancing with routers configured on each side (no NAT used and servers get the benefit of balancing). This is supported by MikroTik RouterOS.
3. packet based balancing (also known as bonding or trunking) - can mess up tcp with out-of-order packets (also needs NAT if only one router doing it). The only real benifit of this if you are controlling both sides. This is not currently supported by MikroTik RouterOS, but is planned in future versions.
4. real bonding of things like synchronous links - needs hardware that supports this on both sides of the link. This is not currently supported by MikroTik RouterOS.

Also, be warned that one download session will never see double the speed. Of course if you had some kind of accelerator on your computer that opened multiple sessions, then you would see the benefit.

Gateway fail-over

The most simple way to do this is to use netwatch. Here we will ping once in 5 seconds the "primary" gateway (2.2.2.2) of the router, and if does not respond, we will switch to the "backup" gateway (3.3.3.1):

```
/system script add name=down source={/ip route \
{... set [/ip route find dst-address=0.0.0.0] gateway 3.3.3.1}
/system script add name=up source={/ip route \
{... set [/ip route find dst-address=0.0.0.0] gateway 2.2.2.1}
/tool netwatch add host=2.2.2.2 interval=5s up-script=up down-script=down
```

Load-balancing 1

Given that we have two Internet providers, each of them have given us an IP address range. We have an internal network and we want to provide load-balancing for this network. To do this, we should set up a default route with both Internet gateways:

```
/ip route add gateway=1.1.1.1,2.2.2.1
```

It is required to make a source NAT (or masquerading) for the outgoing traffic depending on what gateway will it use, or else we will not benefit from having two gateways.

Servers will not get any benefit of load-balancing as there is a predetermined routing path to their addresses unless You can make clients use several addresses of the same server, which are reachable by different paths. This could be done with so called DNS-based load-balancing, when You specify more than one IP address for the same DNS name. This topic will not be covered by RouterOS documentation.

As it is very desirable to provide fail-over together with load-balancing, you can use the following script to maintain the correct gateway list:

```
/system script add name=fo source={
  :local R1
  :local R2
  :if ([/tool netwatch get R1 status]=up) do={:set R1 1.1.1.1}
  :if ([/tool netwatch get R2 status]=up) do={:set R2 2.2.2.1}
   /ip route set [/ip route find dst-address=0.0.0.0/0] \
  gateway=($R1 . , . $R2)
  }
/tool netwatch add comment=R1 host=1.1.1.1 interval=5s up-script=fo \
  down-script=fo
/tool netwatch add comment=R2 host=2.2.2.1 interval=5s up-script=fo \
  down-script=fo
```

We can also expand this script to three or even more gateways. For example, if we had a third gateway of 3.3.3.1, the script would look like this:

```
/system script add name=fo source={
  :local R1
  :local R2
  :local R3
  :if ([/tool netwatch get R1 status]=up) do={:set R1 1.1.1.1}
  :if ([/tool netwatch get R2 status]=up) do={:set R2 2.2.2.1}
  :if ([/tool netwatch get R3 status]=up) do={:set R3 3.3.3.1}
   /ip route set [/ip route find dst-address=0.0.0.0/0] \
  gateway=($R1 . , . $R2 . , . $R3)
  }
/tool netwatch add comment=R1 host=1.1.1.1 interval=5s up-script=fo \
  down-script=fo
/tool netwatch add comment=R2 host=2.2.2.1 interval=5s up-script=fo \
  down-script=fo
/tool netwatch add comment=R3 host=3.3.3.1 interval=5s up-script=fo \
  down-script=fo
```

Load-balancing 2

To do this, you need two RouterOS boxes connected with at least two lines. Let's call R1 the box that is providing Internet to your local users, and R2 that is connected at the other side of the multiple link, and that is connected to your Internet provider(s).

In R1, this is done in the same manner as in the previeous example, just without SRC-NAT (you can make a SRC-NAT on the other one if you want, but this is not required for the method to work). Gateways would be the addresses of R2 put each on a different interface (link).

In R2 you will need to make the same multipath routing as in R1, changing the gateways to the respective addresses of the R1 ends of the lines. Also you will need to change the dst-address from the 0.0.0.0 to the respective network you have behind the R1.

VRRP Script

The following script is working with VRRP protocol. The script is checking '/ip vrrp' submenu, if there could be found some entry with a Backup flag then the script takes the interface names of all entries with a master flag and disables these interfaces under '/interface' list; if the state of all entries is master, then the script takes a name of such an entry and enables it under '/interface' list.

```
:global tmp;:global t;:global iface;:global bool
:set bool 0
:foreach i in [/ip vrrp find backup=yes] do={
  /ip vrrp {
    :set bool 1
    :foreach e in [find master=yes] do={
      :set iface [get $e interface]
      /interface disable [/inter find name=$iface]
      }
    }
  }

:if ($bool = 0) do={
  /ip vrrp {
    :foreach e in [find invalid=true] do={
      :set iface [get $e interface]
      /interface enable [/interf find name=$iface]
      }
    }
  }
```

Application Examples

Registration Table Checking

The following example script checks the wireless registration table for a certain MAC address and if that address is not available, checks for a second MAC address and in case this one is not available too disables the wireless interface, after that enables the wireless interface.

```
/system script add name=search source={
  :if ([/interface wireless reg find mac-address=00:01:02:03:04:05] = 0) do={
    :if ([/interface wireless reg find mac-address=AA:AB:AA:AA:AA:AA] = 0) do={
      /interface wireless disable [find]
      /interface wireless enable [find]
    }
  }
}
```

Cleaning up the firewall

This example will remove everythig from the firewall:

```
:foreach  i in  [/ip firewall find] do={
  /ip firewall rule $i remove [/ip firewall rule $i find]
}
:foreach  i in  [/ip firewall find] do={
        /ip firewall remove $i
}
```

Converting ARP entries from dynamic to static

To convert all the dynamic ARP entries on **ether1** interface to static ones:

```
:foreach i in [/ip arp find dynamic=yes interface=ether1] do={
  /ip arp add copy-from=$i
}
```

Night policy

We can define different rules for different time of the day. For example, here are two scripts that will change the queue rule Cust0. Every day at 9AM the queue will be set to 64Kbit/s and at 5PM the queue will be set to 128Kbit/s.

```
[admin@MikroTik] queue simple> add name=Cust0 interface=public \
\... target-address=192.168.0.0/24 max-limit=64000/64000
[admin@MikroTik] queue simple> /system script
[admin@MikroTik] system script> add name=start_limit source={
  /queue simple set Cust0 max-limit=64000/64000
  }
[admin@MikroTik] system script> add name=stop_limit source={
  /queue simple set Cust0 max-limit=128000/128000
  }
[admin@MikroTik] system script> .. scheduler
[admin@MikroTik] system scheduler> add interval=24h name="set-64k" \
\... start-time=9:00:00 on-event=start_limit
[admin@MikroTik] system scheduler> add interval=24h name="set-128k" \
\... start-time=17:00:00 on-event=stop_limit
```

Another similar example is just to disable all queues at night:

```
[admin@MikroTik] system script> add name=start_limit source={
  /queue simple enable [/queue simple find]
  }
[admin@MikroTik] system script> add name=stop_limit source={
  /queue simple disable [/queue simple find]
  }
[admin@MikroTik] system script> .. scheduler
[admin@MikroTik] system scheduler> add interval=24h name="start" \
\... start-time=9:00:00 on-event=start_limit
[admin@MikroTik] system scheduler> add interval=24h name="stop" \
\... start-time=17:00:00 on-event=stop_limit
```

And here we will reference entries using **comment** property:

```
/ip firewall rule forward add p2p=all-p2p action drop comment=p2p
/system script add name=p2p_disable source={
  /ip firewall rule forward {
    disable [find comment=p2p]
    }
  }
/system script add name=p2p_enable source={
  /ip firewall rule forward {
    enable [find comment=p2p]
    }
  }
/system scheduler add interval=24h name="enable_p2p" \
  start-time=20:00:00 on-event=p2p_enable
/system scheduler add interval=24h name="disable_p2p" \
  start-time=7:00:00 on-event=p2p_disable
```

Managing HotSpot users

To erase a user when his/her total uptime has reached the respective uptime limit:

```
/ system script add name="delete_user" source={
  :foreach i in [/ip hotspot user find profile=default] do={
    :if ([/ip hotspot user get $i limit-uptime]<=[/ip hotspot user get $i \
    uptime]) do={
      /ip hotspot user remove $i
      }
    }
  }
```

Send an e-mail about such a user:

```
/ system script add name="expired_user" source={
  :foreach i in [/ip hotspot user find] do={
    :if ([/ip hotspot user get $i limit-uptime]<=[/ip hotspot user get $i \ uptime])
do={
      /tool e-mail send \
        to=example@example.com \
        subject="Expired user" \
        body=("The account for user '" . [ip hotspot user get $i name] . "' \
\... has been expired. Total uptime is " . [/ip hotspot user get $i uptime])
      }
    }
  }
```

Send an e-mail each time a new user loggs in:

```
/system script add name="new_user" source={
  :foreach a in [/ip hotspot active find] do={
    :if ([/ip hotspot active get $a uptime]<1m) do={
      /tool e-mail send \
        to=example@example.com \
        subject=("'" . [/ip hotspot active get $a user] . "' logged in") \
        body=("User:" . [/ip hotspot active get $a user] . "  Date:" . \
          [/system clock get time] . "   IP address:" . \
          [/ip hotspot active get $a address] . "  MAC address:" . \
```

```
          [/ip hotspot active get $a mac-address] . "  Session timeout:" . \
          [/ip hotspot active get $a session-timeout])
      }
    }
  }
```

Last script assumes that you are running it from the system scheduler with intervals of 1 minute.

Wait for interfaces to disappear

This example reboots the router every minute, but only in case 2 ethernet interfaces were found. If more or less than 2 ethernet interfaces were found, the script makes a support output file and stops the scheduler.

```
/system script add name=disappear source={
  :local cnt
  :set cnt 0
  :foreach i in [/interface find] do={
    :if ([/interface get $i type] = ether) do={
      :incr cnt
      }
    }
  :if ($cnt = 2) do={
    :if ([/system resource get uptime] > 1m) do={
      /system reboot
      }
    } \
    else={
      /system scheduler disable [/system scheduler find]
      /system sup-output
      }
  }
/system scheduler add name=sup on-event=disappear interval=1m
```

Maintaining a Demo user

Suppose we want to give out a demonstration account, and need to maintaint its password intact. We can also check if somebody had not accidently deleted this user, and recreate it if needed.

```
/ system script add name="demo_user" source={
  :global nam
  :global temp
  :set temp 0
  :foreach e in [/user find] do={
    :set nam [/user get $e name]
    :if ($nam="demo") do={
      /user set $e password=""
      :incr temp
      }
    }
  :if ($temp=0) do={
    /user add name=demo group=demo
    }
  }
```

No we can put this in scheduler.

Traffic limiting per amount downloaded

For the purposes of this example we are assuming that you have already added rules with **action=passthrough** and **comment=userN**, where **N** will be in range from 1 to *user-count* to the **forward** chain of your router. Additionally, no other rules should be present in the **forward** chain (you can make your own chain if you need to).

```
:local sum; :local traf;
:set sum 0
/ip firewall rule forward {
  :foreach i in [find] do={:incr sum}
  :for i from=1 to=$sum  do={
    :set traf [get [find comment=("user" . $i)]
bytes]
    :set traf ($traf/1073741824)
    :if ($traf>1) do={:log facility=System-Info message=("user" . $i .
" exceeded 1Gb limit!")}
  }
}
```

You need to run this script from system scheduler to make regular reports.