nft

nft — Administration tool of the nftables framework for packet filtering and classification

Synopsis

nft [**-nNscae**] [**-I** *directory*] [**-f** *filename* | **-i** | *cmd* . . .]
nft -h

nft -v

DESCRIPTION

nft is the command line tool used to set up, maintain and inspect packet filtering and classification rules in the Linux kernel, in the nftables framework. The Linux kernel subsystem is known as nf_tables, and 'nft' stands for Netfilter.

OPTIONS

For a full summary of options, run **nft --help**.

-h , --help
Show help message and all options.

-v , --version
Show version.

-n , --numeric
Show data numerically. When used once (the default behaviour), skip lookup of addresses to symbolic names. Use twice to also show Internet services (port numbers) numerically. Use three times to also show protocols and UIDs/GIDs numerically.

-s , --stateless
Omit stateful information of rules and stateful objects.

-l , --literal
Translate numeric to literal. When used once (the default behav- iour), print services (instead of numerical port numbers). Use twice to perform the IP address to name lookup, this usually re- quires network traffic for DNS lookup that slows down the rule- set listing.

-c , --check
Check commands validity without actually applying the changes.

-a , --handle
Show object handles in output.

-e , --echo
When inserting items into the ruleset using add, insert or replace commands, print notifications just like nft monitor.

-I , --includepath directory
Add the directory *directory* to the list of directories to be searched for included files. This op- tion may be specified multiple times.

-f , --file *filename*
Read input from *filename*. If *filename* is -, read from stdin.
nft scripts must start #!/usr/sbin/nft -f

-i , --interactive
Read input from an interactive readline CLI. You can use quit to exit, or use the EOF marker, nor- mally this is CTRL-D.

INPUT FILE FORMATS

LEXICAL CONVENTIONS

Input is parsed line-wise. When the last character of a line, just before the newline character, is a non-quoted backslash (\), the next line is treated as a continuation. Multiple commands on the same line can be separated using a semicolon (;).

A hash sign (#) begins a comment. All following characters on the same line are ignored.

Identifiers begin with an alphabetic character (a-z,A-Z), followed zero or more alphanumeric characters (a-z,A-Z,0-9) and the characters slash (/), backslash (\), underscore (_) and dot (.). Identifiers using different characters or clashing with a keyword need to be enclosed in double quotes (").

INCLUDE FILES

include *filename*

Other files can be included by using the **include** statement. The directories to be searched for include files can be specified using the **-I/--includepath** option. You can override this behaviour either by prepending ‘./’ to your path to force inclusion of files located in the current working directory (i.e. relative path) or / for file location expressed as an absolute path.

If **-I/--includepath** is not specified, then nft relies on the default directory that is specified at compile time. You can retrieve this default directory via **-h/--help** option.

Include statements support the usual shell wildcard symbols (*,?,[]). Having no matches for an include statement is not an error, if wildcard symbols are used in the include statement. This allows having potentially empty include directories for statements like **include "/etc/firewall/rules/"**. The wildcard matches are loaded in alphabetical order. Files beginning with dot (.) are not matched by include statements.

SYMBOLIC VARIABLES

define variable *expr* **$variable**

Symbolic variables can be defined using the **define** statement. Variable references are expressions and can be used initialize other variables. The scope of a definition is the current block and all blocks contained within.

Using symbolic variables

```
define int_if1 = eth0
define int_if2 = eth1
define int_ifs = { $int_if1, $int_if2 }

filter input iif $int_ifs accept
```

ADDRESS FAMILIES

Address families determine the type of packets which are processed. For each address family the kernel contains so called hooks at specific stages of the packet processing paths, which invoke nftables if rules for these hooks exist.

ip	IPv4 address family.
ip6	IPv6 address family.

inet	Internet (IPv4/IPv6) address family.
arp	ARP address family, handling IPv4 ARP packets.
bridge	Bridge address family, handling packets which traverse a bridge device.
netdev	Netdev address family, handling packets from ingress.

All nftables objects exist in address family specific namespaces, therefore all identifiers include an address family. If an identifier is specified without an address family, the **ip** family is used by default.

IPV4/IPV6/INET ADDRESS FAMILIES

The IPv4/IPv6/Inet address families handle IPv4, IPv6 or both types of packets. They contain five hooks at different packet processing stages in the network stack.

Table 1: IPv4/IPv6/Inet address family hooks

Hook	Description
prerouting	All packets entering the system are processed by the prerouting hook. It is invoked before the routing process and is used for early filtering or changing packet attributes that affect routing.
input	Packets delivered to the local system are processed by the input hook.
forward	Packets forwarded to a different host are processed by the forward hook.
output	Packets sent by local processes are processed by the output hook.
postrouting	All packets leaving the system are processed by the postrouting hook.

ARP ADDRESS FAMILY

The ARP address family handles ARP packets received and sent by the system. It is commonly used to mangle ARP packets for clustering.

Table 2: ARP address family hooks

Hook	Description
input	Packets delivered to the local system are processed by the input hook.
output	Packets send by the local system are processed by the output hook.

BRIDGE ADDRESS FAMILY

The bridge address family handles ethernet packets traversing bridge devices.

The list of supported hooks is identical to IPv4/IPv6/Inet address families above.

NETDEV ADDRESS FAMILY

The Netdev address family handles packets from ingress.

Table 3: Netdev address family hooks

Hook	Description
ingress	All packets entering the system are processed by this hook. It is invoked before layer 3 protocol handlers and it can be used for early filtering and policing.

RULESET

{list | flush} **ruleset** [*family*]

{export} [**ruleset**] {*format*}

The **ruleset** keyword is used to identify the whole set of tables, chains, etc. currently in place in kernel. The following **ruleset** commands exist:

list
Print the ruleset in human-readable format.

flush
Clear the whole ruleset. Note that unlike iptables, this will remove all tables and whatever they contain, effectively leading to an empty ruleset - no packet filtering will happen anymore, so the kernel accepts any valid packet it receives.

export
Print the ruleset in machine readable format. The mandatory *format* parameter may be either **xml** or **json**.

It is possible to limit **list** and **flush** to a specific address family only. For a list of valid family names, see the section called "ADDRESS FAMILIES" above.

Note that contrary to what one might assume, the output generated by **export** is not parseable by **nft -f**. Instead, the output of **list** command serves well for that purpose.

TABLES

{add | create} **table** [*family*] *table* [{flags *flags*}]
{delete | list | flush} **table** [*family*] *table* delete **table** [*family*] handle *handle*

Tables are containers for chains, sets and stateful objects. They are identified by their address family and their name. The address family must be one of **ip**, **ip6**, **inet**, **arp**, **bridge**, **netdev**. The **inet** address family is a dummy family which is used to create hybrid IPv4/IPv6 tables. The **meta expression nfproto** keyword can be used to test which family (ipv4 or ipv6) context the packet is being processed in. When no address family is specified, **ip** is used by default.

add
Add a new table for the given family with the given name.

delete
Delete the specified table.

list
List all chains and rules of the specified table.

flush
Flush all chains and rules of the specified table.

CHAINS

{add | create} **chain** [*family*] *table chain* [{ {*type*} {*hook*} [*device*] {*priority*} } [*policy*]]
{delete | list | flush} **chain** [*family*] {*table*} {*chain*}

Chains are containers for rules. They exist in two kinds, base chains and regular chains. A base chain is an entry point for packets from the networking stack, a regular chain may be used as jump target and is used for better rule organization.

add
Add a new chain in the specified table. When a hook and priority value are specified, the chain is created as a base chain and hooked up to the networking stack.

create
Similar to the **add** command, but returns an error if the chain already exists.

delete
Delete the specified chain. The chain must not contain any rules or be used as jump target.

rename
Rename the specified chain.

list
List all rules of the specified chain.

flush
Flush all rules of the specified chain.

For base chains, **type**, **hook** and **priority** parameters are mandatory.

Table 4: Supported chain types

Type	Families	Hooks	Description
filter	all	all	Standard chain type to use in doubt.
nat	ip, ip6	prerouting, input, output, postrouting	Chains of this type perform Native Address Translation based on conntrack entries. Only the first packet of a connection actually traverses this chain - its rules usually define details of the created conntrack entry (NAT statements for instance).

Table 4: (continued)

Type	Families	Hooks	Description
route	ip, ip6	output	If a packet has traversed a chain of this type and is about to be accepted, a new route lookup is performed if relevant parts of the IP header have changed. This allows to e.g. implement policy routing selectors in nftables.

+ +Apart from the special cases illustrated above (e.g. **nat** type not supporting **+forward** hook or **route** type only supporting **output** hook), there are two +further quirks worth noticing:

+ +* **netdev** family supports merely a single combination, namely **filter** type + and **ingress** hook. Base chains in this family also require the **device** + parameter to be present since they exist per incoming interface only. +* **arp** family supports only **input** and **output** hooks, both in chains of + type **filter**.

The **priority** parameter accepts a signed integer value which specifies the order in which chains with same **hook** value are traversed. The ordering is ascending, i.e. lower priority values have precedence over higher ones.

Base chains also allow to set the chain's **policy**, i.e. what happens to packets not explicitly accepted or refused in contained rules. Supported policy values are **accept** (which is the default) or **drop**.

RULES

[add | insert] **rule** [*family*] {*table*} {*chain*} [position *position*] {*statement*}...

Rules are constructed from two kinds of components according to a set of grammatical rules: expressions and statements.

add
Add a new rule described by the list of statements. The rule is appended to the given chain unless a position is specified, in which case the rule is appended to the rule given by the position.

insert
Similar to the **add** command, but the rule is prepended to the beginning of the chain or before the rule at the given position.

delete
Delete the specified rule.

SETS

{delete | list | flush} **set** [*family*] {*table*} {*set*}
{add | delete} **element** [*family*] {*table*} {*set*} { {*elements*} }

Sets are elements containers of an user-defined data type, they are uniquely identified by an user-defined name and attached to tables.

add
Add a new set in the specified table.

delete
Delete the specified set.

list
Display the elements in the specified set.

flush
Remove all elements from the specified set.

add element
Comma-separated list of elements to add into the specified set.

delete element
Comma-separated list of elements to delete from the specified set.

Table 5: Set specifications

Keyword	Description	Type
type	data type of set elements	string: ipv4_addr, ipv6_addr, ether_addr, inet_proto, inet_service, mark
flags	set flags	string: constant, interval, timeout
timeout	time an element stays in the set	string, decimal followed by unit. Units are: d, h, m, s
gc-interval	garbage collection interval, only available when timeout or flag timeout are active	string, decimal followed by unit. Units are: d, h, m, s
elements	elements contained by the set	set data type
size	maximun number of elements in the set	unsigned integer (64 bit)
policy	set policy	string: performance [default], memory

MAPS

{delete | list | flush} **map** [*family*] {*table*} {*map*}
{add | delete} **element** [*family*] {*table*} {*map*} { {*elements*} }

Maps store data based on some specific key used as input, they are uniquely identified by an user-defined name and attached to tables.

add
Add a new map in the specified table.

delete
Delete the specified map.

list
Display the elements in the specified map.

flush
Remove all elements from the specified map.

add element
Comma-separated list of elements to add into the specified map.

delete element
Comma-separated list of element keys to delete from the specified map.

Table 6: Map specifications

Keyword	Description	Type
type	data type of map elements	string *:* string: ipv4_addr, ipv6_addr, ether_addr, inet_proto, inet_service, mark, counter, quota. Counter and quota can't be used as keys
	flags	map flags
string: constant, interval		elements
elements contained by the map	map data type	
size	maximun number of elements in the map	unsigned integer (64 bit)
	policy	map policy

STATEFUL OBJECTS

{add | delete | list | reset} **type** [*family*] {*table*} {*object*}

Stateful objects are attached to tables and are identified by an unique name. They group stateful information from rules, to reference them in rules the keywords "type name" are used e.g. "counter name".

add
Add a new stateful object in the specified table.

delete
Delete the specified object.

list
Display stateful information the object holds.

reset
List-and-reset stateful object.