Linux IPv6 HOWTO (en)

Peter Bieringer

pb at bieringer dot de

Revision History
Revision Release 0.49 2005-10-03 Revised by: PB
See revision history for more
Revision Release 0.48.1 2005-01-15 Revised by: PB
See revision history for more
Revision Release 0.48 2005-01-11 Revised by: PB
See revision history for more
Revision Release 0.47.1 2005-01-01 Revised by: PB
See revision history for more
Revision Release 0.47 2004-08-30 Revised by: PB
See revision history for more
Revision Release 0.46 2004-03-16 Revised by: PB
See revision history for more

The goal of the Linux IPv6 HOWTO is to answer both basic and advanced questions about IPv6 on the Linux operating system. This HOWTO will provide the reader with enough information to install, configure, and use IPv6 applications on Linux machines.

Table of Contents
1. General
   1.1. Copyright, license and others
   1.2. Category
   1.3. Version, History and To-Do
   1.4. Translations
   1.5. Technical
   1.6. Preface
   1.7. Used terms, glossary and shortcuts
   1.8. Requirements for using this HOWTO
2. Basics
   2.1. What is IPv6?
   2.2. History of IPv6 in Linux
   2.3. What do IPv6 addresses look like?
   2.4. FAQ (Basics)
3. Address types
   3.1. Addresses without a special prefix
   3.2. Network part, also known as prefix
   3.3. Address types (host part)
   3.4. Prefix lengths for routing
4. IPv6-ready system check
4.1. IPv6-ready kernel
4.2. IPv6-ready network configuration tools
4.3. IPv6-ready test/debug programs
4.4. IPv6-ready programs
4.5. IPv6-ready client programs (selection)
4.6. IPv6-ready server programs
4.7. FAQ (IPv6-ready system check)
5. Configuring interfaces
   5.1. Different network devices
   5.2. Bringing interfaces up/down
6. Configuring IPv6 addresses
   6.1. Displaying existing IPv6 addresses
   6.2. Add an IPv6 address
   6.3. Removing an IPv6 address
7. Configuring normal IPv6 routes
   7.1. Displaying existing IPv6 routes
   7.2. Add an IPv6 route through a gateway
   7.3. Removing an IPv6 route through a gateway
   7.4. Add an IPv6 route through an interface
   7.5. Removing an IPv6 route through an interface
   7.6. FAQ for IPv6 routes
8. Neighbor Discovery
   8.1. Displaying neighbors using “ip”
   8.2. Manipulating neighbors table using “ip”
9. Configuring IPv6-in-IPv4 tunnels
   9.1. Types of tunnels
   9.2. Displaying existing tunnels
   9.3. Setup of point-to-point tunnel
   9.4. Setup of 6to4 tunnels
10. Configuring IPv4-in-IPv6 tunnels
11. Kernel settings in /proc/filesystem
   11.1. How to access the /proc/filesystem
   11.2. Entries in /proc/sys/net/ipv6/
   11.3. IPv6-related entries in /proc/sys/net/ipv4/
   11.4. IPv6-related entries in /proc/net/
12. Netlink-Interface to kernel
13. Network debugging
   13.1. Server socket binding
   13.2. Examples for tcpdump packet dumps
14. Support for persistent IPv6 configuration in Linux distributions
   14.1. Red Hat Linux and “clones”
   14.2. SuSE Linux
   14.3. Debian Linux
15. Auto-configuration and mobility
   15.1. Stateless auto-configuration
   15.2. Stateful auto-configuration using Router Advertisement Daemon (radvd)
   15.3. Dynamic Host Configuration Protocol v6 (DHCPv6)
   15.4. Mobility
16. Firewalling
   16.1. Firewalling using netfilter6
3.1. Addresses without a special prefix

3.1.1. Localhost address

This is a special address for the loopback interface, similar to IPv4 with its "127.0.0.1". With IPv6, the localhost address is:

```
0000:0000:0000:0000:0000:0000:0000:0001
```
or compressed:

::1

Packets with this address as source or destination should never leave the sending host.

3.1.2. Unspecified address

This is a special address like "any" or "0.0.0.0" in IPv4. For IPv6 it's:

0000:0000:0000:0000:0000:0000:0000:0000

or:

::

These addresses are mostly used/seen in socket binding (to any IPv6 address) or routing tables.

Note: the unspecified address cannot be used as destination address.

3.1.3. IPv6 address with embedded IPv4 address

There are two addresses which contain an IPv4 address.

3.1.3.1. IPv4-mapped IPv6 address

IPv4-only IPv6-compatible addresses are sometimes used/shown for sockets created by an IPv6-enabled daemon, but only binding to an IPv4 address.

These addresses are defined with a special prefix of length 96 (a.b.c.d is the IPv4 address):

0:0:0:0:ffff:a.b.c.d/96

or in compressed format

::ffff:a.b.c.d/96

For example, the IPv4 address 1.2.3.4 looks like this:

::ffff:1.2.3.4
3.1.3.2. IPv4-compatible IPv6 address

Used for automatic tunneling (RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers), which is being replaced by 6to4 tunneling.

```
0:0:0:0:0:0:a.b.c.d/96
```

or in compressed format

```
::a.b.c.d/96
```

---

3.2. Network part, also known as prefix

Designers defined some address types and left a lot of scope for future definitions as currently unknown requirements arise. RFC 2373 [July 1998] / IP Version 6 Addressing Architecture defines the current addressing scheme but there is already a new draft available: draft-ietf-ipngwg-addr-arch-*txt.

Now lets take a look at the different types of prefixes (and therefore address types):

### 3.2.1. Link local address type

These are special addresses which will only be valid on a link of an interface. Using this address as destination the packet would never pass through a router. It's used for link communications such as:

- anyone else here on this link?
- anyone here with a special address (e.g. looking for a router)?

They begin with (where "x" is any hex character, normally "0")

```
fe8x: <- currently the only one in use.
fe9x:
feax:
febx:
```
An address with this prefix is found on each IPv6-enabled interface after stateless auto-configuration (which is normally always the case).

### 3.2.2. Site local address type

These are addresses similar to the RFC 1918 / Address Allocation for Private Internets in IPv4 today, with the added advantage that everyone who use this address type has the capability to use the given 16 bits for a maximum number of 65536 subnets. Comparable with the 10.0.0.0/8 in IPv4 today.

Another advantage: because it's possible to assign more than one address to an interface with IPv6, you can also assign such a site local address in addition to a global one.

It begins with:

```plaintext
 fecx: <- most commonly used.
 fedx:    
 feex:    
 fexx:    
```

(where "x" is any hex character, normally "0")

Note that there are discussions going on in deprecating this kind of addresses because there are several issues. Read the current draft for more: draft-ietf-ipv6-deprecate-site-local-XY.txt.

For test in labs, such addresses are still a good choice in my humble opinion.

### 3.2.3. Global address type "(Aggregatable) global unicast"

Today, there is one global address type defined (the first design, called "provider based," was thrown away some years ago RFC 1884 / IP Version 6 Addressing Architecture [obsolete], you will find some remains in older Linux kernel sources).

It begins with (x are hex characters)

```plaintext
 2xxx:    
 3xxx:    
```

Note: the prefix "aggregatable" is thrown away in current drafts. There are some further subtypes defined, see below:

#### 3.2.3.1. 6bone test addresses

These were the first global addresses which were defined and in use. They all start with
3ffe:

Example:

3ffe:ffff:100:f102::1

A special 6bone test address which will be never be globally unique begins with

3ffe:ffff:

and is mostly shown in older examples, because if real addresses are shown, its possible for someone to do a copy & paste to their configuration files. Thus inadvertently causing duplicates on a globally unique address. This would cause serious problems for the original host (e.g. getting answer packets for request that were never sent). Because IPv6 is now in production, this prefix will no longer be delegated and probably removed routing after 6.6.2006 (see RFC 3701 / 6bone Phaseout for more).

3.2.3.2. 6to4 addresses

These addresses, designed for a special tunneling mechanism [RFC 3056 / Connection of IPv6 Domains via IPv4 Clouds and RFC 2893 / Transition Mechanisms for IPv6 Hosts and Routers], encode a given IPv4 address and a possible subnet and begin with

2002:

For example, representing 192.168.1.1/5:

2002:c0a8:0101:5::1

A small shell command line can help you generating such address out of a given IPv4 one:

```
ipv4="1.2.3.4"; sla="5"; printf "2002:%02x%02x:%02x%02x:%04x::1" `echo $ipv4
| tr "." "\"` $sla
```

See also tunneling using 6to4 and information about 6to4 relay routers.

3.2.3.3. Assigned by provider for hierarchical routing

These addresses are delegated to Internet service providers (ISP) and begin with

2001:
Prefixes to major (backbone owning) ISPs (also known as LIRs) are delegated by local registries and currently they got a prefix with length 32 assigned.

Any ISP customer can get a prefix with length 48.

3.2.3.4. Addresses reserved for examples and documentation

Currently, two address ranges are reserved for examples and documentation:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3ffe:ffff::/32</td>
<td>EXAMPLENET-WF</td>
</tr>
<tr>
<td>2001:0db8::/32</td>
<td></td>
</tr>
</tbody>
</table>

These address ranges should be filtered based on source addresses and should NOT be routed on border routers to the internet, if possible.

3.2.4. Multicast addresses

Multicast addresses are used for related services.

They always start with \((xx\) is the scope value)

\[ffxy:\]

They are split into scopes and types:

3.2.4.1. Multicast scopes

Multicast scope is a parameter to specify the maximum distance a multicast packet can travel from the sending entity.

Currently, the following regions (scopes) are defined:

- \(ffx1\): node-local, packets never leave the node.
- \(ffx2\): link-local, packets are never forwarded by routers, so they never leave the specified link.
- \(ffx5\): site-local, packets never leave the site.
- \(ffx8\): organization-local, packets never leave the organization (not so easy to implement, must be covered by routing protocol).
- \(ffxe\): global scope.
- others are reserved

3.2.4.2. Multicast types

There are many types already defined/reserved (see RFC 2373 / IP Version 6 Addressing Architecture for details). Some examples are:
• All Nodes Address: ID = 1h, addresses all hosts on the local node (ff01:0:0:0:0:0:0:1) or the connected link (ff02:0:0:0:0:0:0:1).
• All Routers Address: ID = 2h, addresses all routers on the local node (ff01:0:0:0:0:0:0:2), on the connected link (ff02:0:0:0:0:0:0:2), or on the local site (ff05:0:0:0:0:0:0:2)

3.2.4.3. Solicited node link-local multicast address

Special multicast address used as destination address in neighborhood discovery, because unlike in IPv4, ARP no longer exists in IPv6.

An example of this address looks like

```
ff02::1:ff00:1234
```

Used prefix shows that this is a link-local multicast address. The suffix is generated from the destination address. In this example, a packet should be sent to address "fe80::1234", but the network stack doesn't know the current layer 2 MAC address. It replaces the upper 104 bits with "ff02:0:0:0:0:1:ff00::104" and leaves the lower 24 bits untouched. This address is now used 'on-link' to find the corresponding node which has to send a reply containing its layer 2 MAC address.

3.2.5. Anycast addresses

Anycast addresses are special addresses and are used to cover things like nearest DNS server, nearest DHCP server, or similar dynamic groups. Addresses are taken out of the unicast address space (aggregatable global or site-local at the moment). The anycast mechanism (client view) will be handled by dynamic routing protocols.

Note: Anycast addresses cannot be used as source addresses, they are only used as destination addresses.

3.2.5.1. Subnet-router anycast address

A simple example for an anycast address is the subnet-router anycast address. Assuming that a node has the following global assigned IPv6 address:

```
3ffe:ffff:100:f101:210:a4ff:fee3:9566/64  <- Node's address
```

The subnet-router anycast address will be created blanking the suffix (least significant 64 bits) completely:

```
3ffe:ffff:100:f101::/64  <- subnet-router anycast address
```

3.3. Address types (host part)
For auto-configuration and mobility issues, it was decided to use the lower 64 bits as host part of the address in most of the current address types. Therefore each single subnet can hold a large amount of addresses.

This host part can be inspected differently:

### 3.3.1. Automatically computed (also known as stateless)

With auto-configuration, the host part of the address is computed by converting the MAC address of an interface (if available), with the EUI-64 method, to a unique IPv6 address. If no MAC address is available for this device (happens e.g. on virtual devices), something else (like the IPv4 address or the MAC address of a physical interface) is used instead.

Consider again the first example

```
```

here,

```
210:a4ff:fee3:9566
```

is the host part and computed from the NIC's MAC address

```
00:10:A4:E3:95:66
```

using the [IEEE-Tutorial EUI-64](https://www.ieee.org/standards) design for EUI-48 identifiers.

### 3.3.1.1. Privacy problem with automatically computed addresses and a solution

Because the "automatically computed" host part is globally unique (except when a vendor of a NIC uses the same MAC address on more than one NIC), client tracking is possible on the host when not using a proxy of any kind.

This is a known problem, and a solution was defined: privacy extension, defined in RFC 3041 / Privacy Extensions for Stateless Address Autoconfiguration in IPv6 (there is also already a newer draft available: [draft-ietf-ipngwg-temp-addresses-*.txt](https://tools.ietf.org/html/draft-ietf-ipngwg-temp-addresses-04)). Using a random and a static value a new suffix is generated from time to time. Note: this is only reasonable for outgoing client connections and isn't really useful for well-known servers.

### 3.3.2. Manually set
For servers it's probably easier to remember simpler addresses, this can also be accommodated. It is possible to assign an additional IPv6 address to an interface, e.g.

```
3ffe:ffff:100:101::1
```

For manual suffixes like "::1" shown in the above example it's required that the 7th most significant bit is set to 0 (the universal/local bit of the automatically generated identifier). Also some other (otherwise unchosen) bit combinations are reserved for anycast addresses, too.

### 3.4. Prefix lengths for routing

In the early design phase it was planned to use a fully hierarchical routing approach to reduce the size of the routing tables maximally. The reasoning behind this approach were the number of current IPv4 routing entries in core routers (> 104 thousand in May 2001), reducing the need of memory in hardware routers (ASIC "Application Specified Integrated Circuit" driven) to hold the routing table and increase speed (fewer entries hopefully result in faster lookups).

Today's view is that routing will be mostly hierarchically designed for networks with only one service provider. With more than one ISP connections, this is not possible, and subject to an issue named multi-homing (infos on multi-homing: drafts-multi6-I Pv6 Multihoming Solutions).

#### 3.4.1. Prefix lengths (also known as "netmasks")

Similar to IPv4, the routable network path for routing to take place. Because standard netmask notation for 128 bits doesn't look nice, designers employed the IPv4 Classless Inter Domain Routing (CIDR, RFC 1519 / Classless Inter-Domain Routing) scheme, which specifies the number of bits of the IP address to be used for routing. It is also called the "slash" notation.

An example:

```
```

This notation will be expanded:

- Network:

```
3ffe:ffff:0100:0000:0000:0000:0000:0000
```
3.4.2. Matching a route

Under normal circumstances (no QoS) a lookup in a routing table results in the route with the most significant number of address bits means the route with the biggest prefix length matches first.

For example if a routing table shows following entries (list is not complete):

<table>
<thead>
<tr>
<th>Destination</th>
<th>Flags</th>
<th>Metric</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>3ffe:ffff:100::/48</td>
<td>::</td>
<td>U 1 0 0 sit1</td>
<td></td>
</tr>
<tr>
<td>2000::/3</td>
<td>::</td>
<td>192.88.99.1 UG 1 0 0 tun6to4</td>
<td></td>
</tr>
</tbody>
</table>

Shown destination addresses of IPv6 packets will be routed through shown device

<table>
<thead>
<tr>
<th>Destination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3ffe:ffff:100:1:2:3:4:5/48</td>
<td>routed through device sit1</td>
</tr>
<tr>
<td>3ffe:ffff:200:1:2:3:4:5/48</td>
<td>routed through device tun6to4</td>
</tr>
</tbody>
</table>

5.1. Different network devices

On a node, there exist different network devices. They can be collected in classes

- Physically bounded, like eth0, tr0
- Virtually existing, like ppp0, tun0, tap0, sit0, isdn0, ippp0

5.1.1. Physically bounded

Physically bounded interfaces like Ethernet or Token-Ring are normal ones and need no special treatment.

5.1.2. Virtually bounded

Virtually bounded interfaces always need special support

5.1.2.1. IPv6-in-IPv4 tunnel interfaces
These interfaces are normally named sitx. The name sit is a shortcut for Simple Internet Transition. This device has the capability to encapsulate IPv6 packets into IPv4 ones and tunnel them to a foreign endpoint.

sit0 has a special meaning and cannot be used for dedicated tunnels.

5.1.2.2. PPP interfaces

PPP interfaces get their IPv6 capability from an IPv6 enabled PPP daemon.

5.1.2.3. ISDN HDLC interfaces

IPv6 capability for HDLC with encapsulation ip is already built-in in the kernel

5.1.2.4. ISDN PPP interfaces

ISDN PPP interfaces (ippp) aren't IPv6 enabled by kernel. Also there are also no plans to do that because in kernel 2.5.+ they will be replaced by a more generic ppp interface layer.

5.1.2.5. SLIP + PLIP

Like mentioned earlier, this interfaces don't support IPv6 transport (sending is OK, but dispatching on receiving don't work).

5.1.2.6. Ether-tap device

Ether-tap devices are IPv6-enabled and also stateless configured. For use, the module "ethertap" has to be loaded before.

5.1.2.7. tun devices

Currently not tested by me.

5.1.2.8. ATM

01/2002: Aren't currently supported by vanilla kernel, supported by USAGI extension

5.1.2.9. Others

Did I forget an interface?...

5.2. Bringing interfaces up/down
Two methods can be used to bring interfaces up or down.

### 5.2.1. Using "ip"

Usage:

```bash
# ip link set dev <interface> up
# ip link set dev <interface> down
```

Example:

```bash
# ip link set dev eth0 up
# ip link set dev eth0 down
```

### 5.2.2. Using "ifconfig"

Usage:

```bash
# /sbin/ifconfig <interface> up
# /sbin/ifconfig <interface> down
```

Example:

```bash
# /sbin/ifconfig eth0 up
# /sbin/ifconfig eth0 down
```

---

### 6.1. Displaying existing IPv6 addresses

First you should check, whether and which IPv6 addresses are already configured (perhaps auto-magically during stateless auto-configuration).

#### 6.1.1. Using "ip"

Usage:

```bash
# /sbin/ip -6 addr show dev <interface>
```

Example for a static configured host:

```bash
# /sbin/ip -6 addr show dev eth0
```
Example for a host which is auto-configured

Here you see some auto-magically configured IPv6 addresses and their lifetime.

```
# /sbin/ip -6 addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
  inet6 fe80::210:a4ff:fee3:9566/10 scope link
  inet6 3ffe:ffff:0:f101::1/64 scope global
  inet6 fec0:0:0:f101::1/64 scope site
```

### 6.1.2. Using "ifconfig"

Usage:

```
# /sbin/ifconfig

Example (output filtered with grep to display only IPv6 addresses). Here you see
different IPv6 addresses with different scopes.

```
# /sbin/ifconfig eth0 |grep "inet6 addr:"
inet6 addr: fe80::210:a4ff:fee3:9566/10 Scope:Link
inet6 addr: 3ffe:ffff:0:f101::1/64 Scope:Global
inet6 addr: fec0:0:0:f101::1/64 Scope:Site
```

## 6.2. Add an IPv6 address

Adding an IPv6 address is similar to the mechanism of "IP ALIAS" addresses in Linux
IPv4 addressed interfaces.

### 6.2.1. Using "ip"

Usage:
# /sbin/ip -6 addr add <ipv6address>/<prefixlength>  dev <interface>

Example:

# /sbin/ip -6 addr add 3ffe:ffff:0:f101::1/64 dev eth0

6.2.2. Using "ifconfig"

Usage:

# /sbin/ifconfig <interface> inet6 add <ipv6address>/<prefixlength>

Example:

# /sbin/ifconfig eth0 inet6 add 3ffe:ffff:0:f101::1/64

6.3. Removing an IPv6 address

Not so often needed, be carefully with removing non existent IPv6 address, sometimes using older kernels it results in a crash.

6.3.1. Using "ip"

Usage:

# /sbin/ip -6 addr del <ipv6address>/<prefixlength>  dev <interface>

Example:

# /sbin/ip -6 addr del 3ffe:ffff:0:f101::1/64 dev eth0

6.3.2. Using "ifconfig"

Usage:

# /sbin/ifconfig <interface> inet6 del <ipv6address>/<prefixlength>
**7.1. Displaying existing IPv6 routes**

First you should check, whether and which IPv6 addresses are already configured (perhaps auto-magically during auto-configuration).

### 7.1.1. Using "ip"

**Usage:**

```
# /sbin/ip -6 route show [dev <device>]
```

**Example:**

```
# /sbin/ip -6 route show dev eth0
3ffe:ffff:0:f101::/64 proto kernel metric 256 mtu 1500 advmss 1440
fe80::/10     proto kernel metric 256 mtu 1500 advmss 1440
ff00::/8      proto kernel metric 256 mtu 1500 advmss 1440
default       proto kernel metric 256 mtu 1500 advmss 1440
```

### 7.1.2. Using "route"

**Usage:**

```
# /sbin/route -A inet6
```

**Example (output is filtered for interface eth0). Here you see different IPv6 routes for different addresses on a single interface.**

```
# /sbin/route -A inet6 | grep -w "eth0"
3ffe:ffff:0:f101::/64 :: UA 256 0 0 eth0 <- Interface route for global
  address
fe80::/10     ::         UA 256 0 0 eth0 <- Interface route for link-local
  address
ff00::/8      ::         UA 256 0 0 eth0 <- Interface route for all
  multicast
  addresses
::/0          ::         UDA 256 0 0 eth0 <- Automatic default route
```
7.2. Add an IPv6 route through a gateway

Mostly needed to reach the outside with IPv6 using an IPv6-enabled router on your link.

7.2.1. Using "ip"

Usage:

```
# /sbin/ip -6 route add <ipv6network>/<prefixlength> via <ipv6address> [dev <device>]
```

Example:

```
# /sbin/ip -6 route add 2000::/3 via 3ffe:ffff:0:f101::1
```

7.2.2. Using "route"

Usage:

```
# /sbin/route -A inet6 add <ipv6network>/<prefixlength> gw <ipv6address> [dev <device>]
```

A device can be needed, too, if the IPv6 address of the gateway is a link local one.

Following shown example adds a route for all currently global addresses (2000::/3) through gateway 3ffe:ffff:0:f101::1

```
# /sbin/route -A inet6 add 2000::/3 gw 3ffe:ffff:0:f101::1
```
7.3. Removing an IPv6 route through a gateway

Not so often needed manually, mostly done by network configure scripts on shutdown (full or per interface)

7.3.1. Using "ip"

Usage:

```bash
# /sbin/ip -6 route del <ipv6network>/<prefixlength> via <ipv6address> [dev <device>]
```

Example:

```bash
# /sbin/ip -6 route del 2000::/3 via 3ffe:ffff:0:f101::1
```

7.3.2. Using "route"

Usage:

```bash
# /sbin/route -A inet6 del <network>/<prefixlength> [dev <device>]
```

Example for removing upper added route again:

```bash
# /sbin/route -A inet6 del 2000::/3 gw 3ffe:ffff:0:f101::1
```

7.4. Add an IPv6 route through an interface

Not often needed, sometimes in cases of dedicated point-to-point links.

7.4.1. Using "ip"

Usage:

```bash
# /sbin/ip -6 route add <ipv6network>/<prefixlength> dev <device>
```
Metric "1" is used here to be compatible with the metric used by route, because the default metric on using "ip" is "1024".

### 7.4.2. Using "route"

**Usage:**

```bash
# /sbin/route -A inet6 add <network>/<prefixlength> dev <device>
```

**Example:**

```bash
# /sbin/route -A inet6 add 2000::/3 dev eth0
```

### 7.5. Removing an IPv6 route through an interface

Not so often needed to use by hand, configuration scripts will use such on shutdown.

#### 7.5.1. Using "ip"

**Usage:**

```bash
# /sbin/ip -6 route del <ipv6network>/<prefixlength> dev <device>
```

**Example:**

```bash
# /sbin/ip -6 route del 2000::/3 dev eth0
```

#### 7.5.2. Using "route"

**Usage:**
Example:

```
# /sbin/route -A inet6 del 2000::/3 dev eth0
```

.6. FAQ for IPv6 routes

7.6.1. Support of an IPv6 default route

One idea of IPv6 was a hierachical routing, therefore only less routing entries are needed in routers.

There are some issues in current Linux kernels:

7.6.1.1. Clients (not routing any packet!)

Client can setup a default route like prefix ":/0", they also learn such route on autoconfiguration e.g. using radvd on the link like following example shows:

```
# ip -6 route show | grep ^default
default via fe80::212:34ff:fe12:3450 dev eth0 proto kernel metric 1024 expires
\ 29sec mtu 1500 advmss 1440
```

7.6.1.2. Routers in case of packet forwarding

Current mainstream Linux kernel (at least <= 2.4.17) don't support default routes. You can set them up, but the route lookup fails when a packet should be forwarded (normal intention of a router).

Therefore at this time "default routing" can be setup using the currently only global address prefix "2000::/3".

The USAGI project already supports this in their extension with a hack.

Note: take care about default routing without address filtering on edge routers. Otherwise unwanted multicast or site-local traffic leave the edge.
through an interface

8.1. Displaying neighbors using "ip"

With following command you can display the learnt or configured IPv6 neighbors

```bash
# ip -6 neigh show [dev <device>]
```

The following example shows one neighbor, which is a reachable router

```bash
# ip -6 neigh show
fe80::201:23ff:fe45:6789 dev eth0 lladdr 00:01:23:45:67:89 router nud reachable
```

8.2. Manipulating neighbors table using "ip"

8.2.1. Manually add an entry

With following command you are able to manually add an entry

```bash
# ip -6 neigh add <IPv6 address> lladdr <link-layer address> dev <device>
```

Example:

```bash
# ip -6 neigh add fec0::1 lladdr 02:01:02:03:04:05 dev eth0
```

8.2.2. Manually delete an entry

Like adding also an entry can be deleted:

```bash
# ip -6 neigh del <IPv6 address> lladdr <link-layer address> dev <device>
```

Example:

```bash
# ip -6 neigh del fec0::1 lladdr 02:01:02:03:04:05 dev eth0
```
8.2.3. More advanced settings

The tool "ip" is less documented, but very strong. See online "help" for more:

```bash
# ip -6 neigh help
Usage: ip neigh { add | del | change | replace } { ADDR [ lladdr LLADDR ]
   [ nud { permanent | noarp | stale | reachable } ]
   [ proxy ADDR ] [ dev DEV ]
   ip neigh {show|flush} [ to PREFIX ] [ dev DEV ] [ nud STATE ]
```

Looks like some options are only for IPv4...if you can contribute information about flags and advanced usage, pls. send.

---

15.1. Stateless auto-configuration

Is supported and seen on the assigned link-local address after an IPv6-enabled interface is up.

Example:

```bash
# ip -6 addr show dev eth0 scope link
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qlen1000
inet6 fe80::211:d8ff:fe6b:f0f5/64 scope link
    valid_lft forever preferred_lft forever
```