Package ‘iptools’

December 9, 2018

Type Package
Title Manipulate, Validate and Resolve 'IP' Addresses
Version 0.6.1
Date 2018-12-08
Author Bob Rudis <bob@rud.is> [aut, cre],
Oliver Keyes <ironholds@gmail.com> [aut],
Tim Smith [ctb]
Maintainer Bob Rudis <bob@rud.is>
Description A toolkit for manipulating, validating and testing 'IP' addresses and
ranges, along with datasets relating to 'IP' addresses. Tools are also provided
to map 'IPv4' blocks to country codes. While it primarily has support for the 'IPv4'
address space, more extensive 'IPv6' support is intended.
License MIT + file LICENSE
URL https://github.com/hrbrmstr/iptools
BugReports https://github.com/hrbrmstr/iptools/issues
NeedsCompilation yes
SystemRequirements C++11
Depends R (>= 3.0.0)
Suggests testthat, knitr, rmarkdown
Imports Rcpp (>= 0.11.2), utils, stats, AsioHeaders, stringi, readr,
triebeard
Encoding UTF-8
LinkingTo BH, Rcpp, AsioHeaders
VignetteBuilder knitr
RoxygenNote 6.0.1.9000
Repository CRAN
Date/Publication 2018-12-09 14:40:02 UTC
asn_table_to_trie

Description

Convert a pyasn generated CIDR data file to a trie

Usage

asn_table_to_trie(asn_table_file)
cached_country_cidrs

Arguments

asn_table_file filename of dat file (can be gzip'd)

Examples

asn_table_to_trie(system.file("test", "rib.tst.gz", package="iptools"))

Description

Inspect cached CIDR countries

Usage
cached_country_cidrs()

---

country_ranges Return CIDR ranges for given ISO 3166-1 alpha-2 country codes

Description

Query http://www.iwik.org/ipcountry/ for the CIDR ranges for a given set of ISO 3166-1 alpha-2 country codes and return the results in a named list of character vectors.

Usage
country_ranges(countries)

Arguments
countries character vector of ISO 3166-1 alpha-2 country codes (case-insensitive)

Details

An internal cache of the CIDR query results are maintained as they only change daily (12:00 CET). The cache can be flushed with flush_country_cidrs and which codes have been cached can be retrieved with cached_country_cidrs).

Value

named list of character vectors of CIDR blocks
Note

This function requires internet connectivity as it pulls daily updated data from http://www.iwik.org/ipcountry/.

Examples

```r
rng <- country_ranges(c("PW", "UZ"))
```

---

**expand_ipv6**

*Expand an IPv6 address from an abbreviated version*

Description

Expand an IPv6 address from an abbreviated version

Usage

```r
expand_ipv6(ip_addresses)
```

Arguments

- `ip_addresses`: a vector of IPv6 IP addresses.

Value

a character vector of expanded IPv6 addresses

References


---

**flush_country_cidrs**

*Flush the country CIDR cache*

Description

Flush the country CIDR cache

Usage

```r
flush_country_cidrs()
```
**get_all_country_ranges**

*Fetch all country CIDR blocks*

**Description**

Iterates through all the country codes and returns a named list of CIDR blocks for all those countries.

**Usage**

```r
get_all_country_ranges()
```

**Value**

named list of character vectors of CIDR blocks

**Note**

This is an expensive operation as it pulls 249 files from `http://www.iwik.org/ipcountry/`. Try not to do this too often.

---

**hilbert_encode**

*Encode an IPv4 address to Hilbert space*

**Description**

Encode an IPv4 address to Hilbert space

**Usage**

```r
hilbert_encode(x, bpp = 8L)
```

**Arguments**

- `x` IPv4 address
- `bpp` Hilbert depth (max 12)
hostname_to_ip

Returns the IP addresses associated with a hostname.

Description

takes in a vector of hostnames and returns the IP addresses from each hostname’s DNS entries. Compatible with both IPv4 and IPv6 addresses.

Usage

hostname_to_ip(hostnames)

Arguments

hostnames a vector of hostnames.

Value

a list the length of hostnames, containing the IP addresses from each hostname’s DNS entries. In the event that a hostname cannot be resolved, the list element will consist of a length-1 character vector containing "Not resolved".

See Also

ip_to_hostname for the opposite functionality - resolving IP addresses to their associated hostname(s) - and ip_to_numeric for converting IP addresses retrieved from hostname_to_ip into their numeric representation.

Examples

## Not run:
# One hostname
hostname_to_ip("dds.ec")
## [1] "162.243.111.4"

# Multiple hostnames
hostname_to_ip(c("dds.ec","ironholds.org"))
## [[1]]
## [1] "162.243.111.4"
## [2]
## [1] "104.131.2.226"

## End(Not run)
**host_count**

Return the number of hosts in a CIDR block

**Description**

TODO: ipv4 validation

**Usage**

```r
cidr = c("10.0.0.0/8", "192.168.0.0/16")
host_count(cidr)
```

**Arguments**

cidr character vector of IPv4 CIDRs

**Examples**

```r
host_count("1.2.3.4/24")
```

---

**iana_assignments**

IANA IPv4 Address Space Registry

**Description**

This dataset contains the registry of address space assignments for IPv4 IP addresses, as set by IANA. It consists of a data.frame containing the columns:

- **prefix**: A block of IPv4 (CIDR notation) addresses that has been registered for a particular purpose (e.g. "100.64.0.0/10")
- **designation**: The entity the block is assigned to.
- **date**: the assignment date of the block, stored as YYYY-MM.
- **whois**: whois registry [whois.afrinic.net|whois.apnic.net|whois.arin.net|whois.lacnic.net|whois.ripe.net]
- **status**: status of the assignment [ALLOCATED|LEGACY|RESERVED]

**Usage**

```r
data(iana_assignments)
```

**Format**

A data frame with 256 rows and 5 variables

**Note**

Last updated 2015-05-01.
References

- The IANA page on the IPv4 assignments.
- RFC1466.

See Also

iana_assignments_refresh for updating the dataset, and iana_special_assignments for particular, special IPv4 assignments.

iana_assignments_refresh

Refresh iptools Internal Datasets

Description

iptools contains a variety of internal datasets. While these should be updated on a regular basis by the package authors and maintainers, they can also be refreshed by you, the user, using this collection of functions. Each one takes the form [dataset_name]_refresh() to make it clear which dataset is updated by which function.

Usage

iana_assignments_refresh()

iana_special_assignments_refresh()

iana_ports_refresh()

Examples

## Not run:

#update iana_assignments

iana_assignments_refresh()

#[1] TRUE

#update iana_special_assignments

iana_special_assignments_refresh()

#[1] TRUE

## End(Not run)
This is the dataset of IANA service names and their assigned ports and transport protocols - along with related metadata.

- **service_name**: The service name for the port assignment
- **port_number**: The ports assigned to that service. This can be individual ports, or a range.
- **transport_protocol**: The transport protocol(s) of the port assignment - [dccp|sctp|tcp|udp]
- **description**: An explanation of the port assignment
- **assignee**: The name of the individual or organisation to whom the assignment is made
- **contact**: The name of the individual or organisation who serves as the contact person for the assignment.
- **registration_date**: The date the assignment was registered on. This may be empty, in the case of early assignments; otherwise, it is stored in the form "YYYY-MM".
- **modification_date**: The date of any modification to the assignment. Same format as registration_date
- **reference**: A description of (or a reference to a document describing) the protocol or application using this port
- **known_unauthorised_uses**: A list of uses by applications or organizations who are not the assignee
- **assignment_notes**: Indications of owner/name change, or any other assignment process issue

**Usage**

`data(iana_ports)`

**Format**

A data frame with 13,659 rows and 12 variables

**Note**

Last updated 2015-05-02

**References**

The IANA list. RFC6335
Description

This dataset contains the registry of special address space assignments for IPv4 IP addresses, as set by IANA. It consists of a data.frame containing the columns:

- **address_block**: the full IPv4 range (chr) (e.g. "11.0.0.0/8")
- **name**: the descriptive name for the special-purpose address block
- **rfc**: the Request for Comment (RfC) through which the special-purpose address block was requested
- **allocation_date**: the allocation date of the block, stored as YYYY-MM.
- **source**: whether an address from the allocated special-purpose address block is valid when used as the source address of an IP datagram that transits two devices (TRUE) or not (FALSE).
- **destination**: whether an address from the allocated special-purpose address block is valid when used as the destination address of an IP datagram that transits two devices (TRUE) or not (FALSE).
- **forwardable**: whether a router may forward an IP datagram whose destination address is drawn from the allocated special-purpose address block between external interfaces (TRUE) or not (FALSE).
- **global**: whether an IP datagram whose destination address is drawn from the allocated special-purpose address block is forwardable beyond a specified administrative domain (TRUE) or not (FALSE).
- **reserved_by_protocol**: whether the special-purpose address block is reserved by IP, itself. This value is TRUE if the RFC that created the special-purpose address block requires all compliant IP implementations to behave in a special way when processing packets either to or from addresses contained by the address block, and FALSE otherwise.

Usage

```r
data(iana_special_assignments)
```

Format

A data frame with 256 rows and 5 variables

Note

Last updated 2014-08-07
ips_in_cidrs

References

- The IANA page on specially assigned blocks.
- RFC5376
- RFC6890

See Also

iana_special_assignments_refresh to refresh this dataset, and iana_assignments for a dataset covering general (non-special) IPv4 assignments.

ips_in_cidrs     Determine if a vector if IPv4 addresses are in a vector of CIDRs

Description

Determine if a vector if IPv4 addresses are in a vector of CIDRs

Usage

ips_in_cidrs(ips, cidrs)

Arguments

ips character vector or numeric vector of IPv4 addresses
cidrs character vector or numeric vector of IPv4 CIDRs

Value

data_frame with ips column and a logical in_cdir column

Note

auto-appends /32 if a bare IPv4 is detected

Examples

ips_in_cidrs(
  c("4.3.2.1", "1.2.3.4", "1.20.113.10", "5.190.145.5"),
  c("5.190.144.0/21", "1.20.113.0/24")
)
iptools  
*A package to quickly and easily handle IP addresses.*

Description

A toolkit for manipulating, validating and testing 'IP' addresses and ranges, along with datasets relating to 'IP' addresses. Tools are also provided to map IPv4 blocks to country codes. While it primarily has support for the 'IPv4' address space, more extensive 'IPv6' support is intended.

ip_classify  
*Identify whether an IP address is IPv4 or IPv6*

Description

Identify the form (IPv4 or IPv6) of a vector of IP addresses. This can also be used to validate IPs.

Usage

```r
ip_classify(ip_addresses)
```

Arguments

- `ip_addresses`  
a vector of IPv4 or IPv6 IP addresses.

Value

a vector containing the class of each input IP address; either "IPv4", "IPv6" or, for IP addresses that were not valid, NA.

See Also

- `is_valid` et al for logical checks of IP addresses, `ip_to_hostname` for resolving IP addresses to their hostnames, and `ip_to_numeric` for converting (IPv4) IP addresses to their numeric representation.

Examples

```r
# IPv4
ip_classify("173.194.123.100")
# [1] "IPv4"

# IPv6
ip_classify("2607:f8b0:4006:80b::1004")
# [1] "IPv6"

# Invalid
```
Description

\texttt{ip\_in\_any} checks whether a vector of IP addresses fall within any of the specified ranges.

Usage

\texttt{ip\_in\_any(ip\_addresses, ranges)}

Arguments

- \texttt{ip\_addresses} character vector of IP addresses
- \texttt{ranges} character vector of CIDR ranges

Value

A logical vector of whether a given IP was in any of the ranges

Examples

```r
## Not run:
north_america <- unlist(country_ranges(countries=c("US", "CA", "MX")))
germany <- unlist(country_ranges("DE"))

set.seed(1492)
targets <- ip_random(1000)

for_sure <- range_generate(sample(north_america, 1))
all(ip\_in\_any(for\_sure, north\_america)) # should be TRUE
## [1] TRUE

absolutely_not <- range_generate(sample(germany, 1))
any(ip\_in\_any(absolutely_not, north\_america)) # should be FALSE
## [1] FALSE

who_knows_na <- ip\_in\_any(targets, north\_america)
who_knows_de <- ip\_in\_any(targets, germany)

sum(who\_knows\_na)
## [1] 464

sum(who\_knows\_de)
## [1] 43

## End(Not run)
```
ip_in_range

Description

ip_in_range checks whether a vector of IP addresses fall within particular IP range(s).

Usage

ip_in_range(ip_addresses, ranges)

Arguments

ip_addresses a vector of IP addresses
ranges either a vector of ranges equal in length to ip_addresses, or a single range. If the former, ip_in_range will compare each IP to the equivalent range. If the latter, each IP will be compared to the single range provided.

Value

a logical vector, where TRUE indicates the relevant IP is in the range, and FALSE indicates that the IP is not in the range, or is an invalid IP address.

See Also

range_boundaries for identifying the minimum and maximum IPs within a range, and validate_range for validating that a range exists.

Examples

# Is this in the range? Yes!
ip_in_range("172.18.0.1","172.18.0.0/28")
#[1] TRUE

ip_numeric_to_binary_string

Convert a character vector of IPv4 addresses to a character vector of bit strings.

Description

Convert a character vector of IPv4 addresses to a character vector of bit strings.
Usage

`ip_numeric_to_binary_string(input)`

Arguments

- `input` numeric vector of IP addresses

Description

`ip_random` generates random IP addresses. These currently only follow IPv4 standards, since IPv6 addresses are too large to be stored in R in their numeric form. All IPs generated this way are valid.

Usage

`ip_random(n)`

Arguments

- `n` the number of IP addresses to randomly generate.

Value

a character vector of randomly-generated IPv4 addresses, in their dotted-decimal form.

See Also

`ip_to_numeric` for converting random_ips' output to its numeric form, and `range_generate` for generating all IP addresses within a specific range.

Examples

```r
ip_random(1)
#> [1] "49.20.57.31"
```
**ip_to_asn**  
*Match IP addresses to autonomous systems*

**Description**
Match IP addresses to autonomous systems

**Usage**
```
ip_to_asn(cidr_trie, ip)
```

**Arguments**
- `cidr_trie`: trie created with `asn_table_to_trie()`
- `ip`: character vector or numeric vector of IPv4 addresses

**Examples**
```
tbl <- asn_table_to_trie(system.file("test", "rib.tst.gz", package="iptools"))
ip_to_asn(tbl, "5.192.0.1")
```

---

**ip_to_binary_string**  
*Convert a numeric vector of IPv4 addresses to a character vector of bit strings.*

**Description**
Convert a numeric vector of IPv4 addresses to a character vector of bit strings.

**Usage**
```
ip_to_binary_string(input)
```

**Arguments**
- `input`: character vector of IP addresses
ip_to_hostname  

Return the hostname associated with particular IP addresses

Description

the opposite, in some ways, of hostname_to_ip, ip_to_hostname consumes a vector of IP addresses and provides a list of the hostnames that those IPs resolve to. Compatible with both IPv4 and IPv6 addresses.

Usage

ip_to_hostname(ip_addresses)

Arguments

ip_addresses a vector of IP addresses.

Value

a list, each entry containing a vector of hostnames for the equivalent input IP address (mostly, this will only be one hostname but not always). If the IP cannot be resolved, the list element will be the string "Invalid IP address".

See Also

hostname_to_ip, for the reverse operation.

Examples

## Not run:
ip_to_hostname("192.168.0.1")
 [[1]]
 [1] "dds.ec"

## End(Not run)

ip_to_numeric  

convert between numeric and dotted-decimal IPv4 forms.

Description

ip_to_numeric takes IP addresses stored in their human-readable representation ("192.168.0.1") and converts it to a numeric representation (3232235521). numeric_to_ip performs the same operation, but in reverse. Due to limitations in R's support for colossally big numbers, this currently only works for IPv4 IP addresses.
Usage

ip_to_numeric(ip_addresses)
numeric_to_ip(ip_addresses)

Arguments

ip_addresses a vector of IP addresses, in their numeric or dotted-decimal form depending on
the function.

Value

For `ip_to_numeric`: a vector containing the numeric representation of `ip_addresses`. If an IP is
invalid (either because it’s an IPv6 address, or isn’t an IP address at all) the returned value for that
IP will be 0.

For `numeric_to_ip`: a vector containing the dotted-decimal representation of `ip_addresses`, as
character strings. If a value cannot be resolved to an IPv4 address, it will appear as "0.0.0.0" or an
empty string.

Examples

#Convert your local, internal IP to its numeric representation.
ip_to_numeric("192.168.0.1")
#[1] 3232235521

#And back
numeric_to_ip(3232235521)
#[1] "192.168.0.1"

Description

Check whether an IP address is valid with `is_valid`, IPv4 with `is_ipv4`, IPv6 with `is_ipv6`, or
multicast (intended to point to multiple machines) with `is_multicast`

Usage

is_multicast(ip_addresses)

is_ipv4(ip_addresses)

is_ipv6(ip_addresses)

is_valid(ip_addresses)
range_boundaries

Arguments

   ip_addresses  a vector of IP addresses

Value

   a vector of TRUE or FALSE values, indicating whether an IP is multicast or not, or NA values if the IP addresses are NAs.

See Also

   ip_classify for character rather than logical classification.

Examples

   # This is multicast
   is_multicast("224.0.0.2")

   # It's also IPv4
   is_ipv4("224.0.0.2")

   # It's not IPv6
   is_ipv6("224.0.0.2")

range_boundaries  calculate the maximum and minimum IPs in an IP range

Description

   when provided with a vector of IP ranges ("172.18.0.0/28"), range_boundaries calculates the maximum and minimum IP addresses in that range.

Usage

   range_boundaries(ranges)

Arguments

   ranges  a vector of IP ranges. Currently only IPv4 ranges work.

Value

   a data.frame of four columns, "minimum_ip" (containing the smallest IP in the provided range) and "maximum_ip" (containing the largest). "min_numeric" & "max_numeric" (the min & max numeric versions of "minimum_ip" and "maximum_ip") and the original range string. If the range was invalid, both columns will contain Invalid as the value.
range_boundaries_to_cidr

Description

takes in a single start/end pair and returns a character vector of all the CIDR blocks necessary to contain the range.

Usage

range_boundaries_to_cidr(ip_start, ip_end)

Arguments

ip_start, ip_end
range start/end (numeric)

Value

character vector

Examples

range_boundaries_to_cidr(ip_to_numeric("192.100.176.0"),
ip_to_numeric("192.100.179.255")
)
## [1] "192.100.176.0/22"
range_generate

generate all IP addresses within a range

Description

generates a vector containing all IP addresses within a provided range. Currently IPv4 only due to R's support (or lack thereof) for really big numbers.

Usage

range_generate(range)

Arguments

range an IPv4 IP range

Value

a character vector containing each IPv4 IP address within the provided range.

See Also

ip_random for randomly-generated IPs, or ip_to_numeric for converting generate_range's output to its numeric form.

Examples

range_generate("172.18.0.0/28")
# [1] "172.18.0.0" "172.18.0.1" "172.18.0.2" "172.18.0.3" "172.18.0.4"
# [6] "172.18.0.5" "172.18.0.6" "172.18.0.7" "172.18.0.8" "172.18.0.9"
# [11] "172.18.0.10" "172.18.0.11" "172.18.0.12" "172.18.0.13" "172.18.0.14" "172.18.0.15"

v6_scope

Return the scope of an IPv6 address (string)

Description

Return the scope of an IPv6 address (string)

Usage

v6_scope(ip_addresses)

Arguments

ip_addresses a vector of IPv6 IP addresses.
validate_range

Value

a numeric vector of scopes

References


validate_range  check whether IPv4 ranges are valid

Description

validate_range checks whether a vector of IPv4 CIDR ranges ("127.0.0.1/32") are valid or not.

Usage

validate_range(ranges)

Arguments

ranges         a vector of IPv4 ranges

Value

a logical vector, where TRUE indicates that the provided entry is valid, and FALSE that it is not (or isn’t an IP range at all)

See Also

ip_classify for classifying (and, incidentally, validating) IPv4 and IPv6 addresses, or range_boundaries for identifying the minimum and maximum IPs within a range.

Examples

validate_range("127.0.0.1/32")
# [1] TRUE
validate_range("127.0.0.1/33")
# [1] FALSE
**xff_extract**

Take vectors of IPs and X-Forwarded-For headers and produce single, normalised IP addresses.

**Description**

xff_extract takes IP addresses and x_forwarded_for values and, in the event that x_forwarded_for is non-null, attempts to extract the "real" IP closest to the client.

**Usage**

xff_extract(ip_addresses, x_forwarded_for)

**Arguments**

- ip_addresses: a vector of IP addresses
- x_forwarded_for: an equally-sized vector of X-Forwarded-For header contents.

**Value**

A vector of IP addresses, incorporating the XFF header value where appropriate.

**Examples**

xff_extract("192.168.0.1", "193.168.0.1, 230.98.107.1")
Index

*Topic datasets
  iana_assignments, 7
  iana_ports, 9
  iana_special_assignments, 10

asn_table_to_trie, 2
cached_country_cidrs, 3
country_ranges, 3
expand_ipv6, 4
flush_country_cidrs, 4
get_all_country_ranges, 5
hilbert_encode, 5
host_count, 7
hostname_to_ip, 6, 17

iana_assignments, 7, 11
iana_assignments_refresh, 8, 8
iana_ports, 9
iana_ports_refresh
  (iana_assignments_refresh), 8
iana_special_assignments, 8, 10
iana_special_assignments_refresh, 11
iana_special_assignments_refresh
  (iana_assignments_refresh), 8
ip_classify, 12, 19, 22
ip_in_any, 13
ip_in_range, 14, 20
ip_numeric_to_binary_string, 14
ip_random, 15, 21
ip_to_asn, 16
ip_to_binary_string, 16
ip_to_hostname, 12, 17
ip_to_numeric, 6, 12, 15, 17, 20, 21
ips_in_cidrs, 11
iptools, 12
iptools-package (iptools), 12

is_ipv4 (is_multicast), 18
is_ipv6 (is_multicast), 18
is_multicast, 18
is_valid, 12
is_valid(is_multicast), 18
numeric_to_ip (ip_to_numeric), 17
range_boundaries, 14, 19, 22
range_boundaries_to_cidr, 20
range_generate, 15, 21

v6_scope, 21
validate_range, 14, 22

xff_extract, 23