Cheatsheet — Strumenti di Rete IP

Reti e Sistemi Distribuiti Mod.B · Università di Messina · roberto.marino@unime.it

ip addr / ip route

Mostra interfacce e indirizzi

```
ip addr show
ip addr show eth0
ip -4 addr show          # solo IPv4
ip -6 addr show          # solo IPv6
```

Lettura output

```
# inet 192.168.1.47/24 brd 192.168.1.255
#    ^^^^ indirizzo/prefix   ^^^ broadcast
# scope global   --> usabile globalmente
# scope link     --> solo link-local
```

Tabella di routing

```
ip route show
# default via 192.168.1.1 dev eth0
#    ^^^^ default gateway
# 192.168.1.0/24 dev eth0
#    ^^^^ consegna diretta su eth0
```

Routing verso una destinazione

```
ip route get 8.8.8.8
# via 192.168.1.1 dev eth0
# --> consegna INDIRETTA (usa gateway)

ip route get 192.168.1.5
# dev eth0 src 192.168.1.47
# --> consegna DIRETTA (stessa subnet)
```

▷ **Consegna diretta**: dst AND mask == src AND mask. **Consegna indiretta**: risultati diversi, pacchetto al gateway. L'IP dst *non cambia* mai lungo il percorso; il MAC sì.

ipcalc / ipcalc-ng

Analisi di una subnet

```
ipcalc 192.168.1.47/24
# Address:   192.168.1.47
# Netmask:   255.255.255.0 = 24
# Network:   192.168.1.0/24
# HostMin:   192.168.1.1
# HostMax:   192.168.1.254
# Broadcast: 192.168.1.255
# Hosts/Net: 254
```

Subnetting: dividere una rete

```
# Dividi 192.168.1.0/24 in /26
ipcalc 192.168.1.0/24 /26
# --> 4 subnet da 62 host ciascuna:
#      192.168.1.0/26    (0-63)
#      192.168.1.64/26   (64-127)
#      192.168.1.128/26  (128-191)
#      192.168.1.192/26  (192-255)
```

ipcalc-ng: funzionalità avanzate

```
# Split in /26 (ipcalc-ng)
ipcalc-ng -S 26 192.168.1.0/24

# Output JSON (per script)
ipcalc-ng -j 192.168.1.0/24

# Info GeoIP su IP pubblico
ipcalc-ng --all-info 8.8.8.8
# Country: United States
# Coordinates: 37.750000,-97.822000

# Verifica validita' indirizzo
ipcalc-ng -c 192.168.1.47
```

[!] `ipcalc` classico (Red Hat/CentOS) non supporta IPv6 e non è più mantenuto. Su Ubuntu/Debian moderni usa `ipcalc-ng`.

Calcolo a mano: host disponibili

```
# host_max = 2^(32-prefix) - 2
# /24 --> 2^8  - 2 = 254
# /25 --> 2^7  - 2 = 126
# /26 --> 2^6  - 2 = 62
# /30 --> 2^2  - 2 = 2    (point-to-point)
# /32 --> 2^0  - 2 = 0    (singolo host)
```

sipcalc

Analisi dettagliata con hex e decimal

```
sipcalc 192.168.0.1/24
# Host address           - 192.168.0.1
# Host address (hex)     - C0A80001
# Host address (dec)     - 3232235521
# Network address        - 192.168.0.0
# Network mask           - 255.255.255.0
# Network mask (hex)     - FFFFFF00
# Broadcast address      - 192.168.0.255
# Cisco wildcard         - 0.0.0.255
# Addresses in network   - 256
# Network range          - 192.168.0.0 - 192.168.0.255
# Usable range           - 192.168.0.1 - 192.168.0.254
```

Analisi IPv6

```
sipcalc 2001:db8::/32
# Network                - 2001:db8::
# Prefix length          - 32
# Network range:
#   2001:0db8:0000::
#   2001:0db8:ffff:...
```

▷ La **Cisco wildcard mask** è il complemento della subnet mask: usata nelle ACL Cisco. Es. mask `255.255.255.0` -> wildcard `0.0.0.255`.

ping / traceroute

ping — raggiungibilità ICMP

```
ping -c 4 8.8.8.8           # 4 pacchetti
ping -c 4 google.com        # risolve DNS
ping -i 0.2 -c 10 host      # ogni 0.2s
ping -s 1400 192.168.1.1    # payload 1400B
ping -W 1 192.168.1.1       # timeout 1s

# Output:
# 64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=12.3 ms
#    ^^^^                                ^^^      ^^^
#    dimensione risposta                 hop rimasti RTT
```

▷ `ping` usa **ICMP Echo Request/Reply**. Lo stesso protocollo gestito con `AF_INET / SOCK_RAW / IPPROTO_ICMP` nei raw socket.

traceroute — percorso dei pacchetti

```
traceroute google.com
traceroute -n google.com     # no DNS lookup
traceroute -m 15 8.8.8.8     # max 15 hop
traceroute -I 8.8.8.8        # usa ICMP (root)
traceroute -T -p 80 host     # usa TCP SYN

# Output:
#  1  192.168.1.1     1.2 ms   (gateway LAN)
#  2  10.x.x.x        8.4 ms   (router ISP)
#  3  * * *                    (hop non risponde)
#  4  8.8.8.8         12.3 ms
```

[!] `* * *` indica che il router intermedio non risponde a ICMP TTL Exceeded — non necessariamente un problema: il pacchetto può comunque passare.

whois

Proprietario di un IP pubblico

```
whois 8.8.8.8
# OrgName:  Google LLC
# NetRange: 8.8.8.0 - 8.8.8.255
# CIDR:     8.8.8.0/24
# OrgRef:   ARIN

whois 150.146.1.1
# inetnum: 150.146.0.0 - 150.146.255.255
# descr:   Universita' di Messina
# source:  RIPE
```

Blocchi storici di classe A

```
whois 17.0.0.1     # Apple Inc.
whois 18.0.0.1     # MIT
whois 19.0.0.1     # Ford Motor Co.
# Assegnati negli anni '80: 16M indirizzi
# ciascuno -- ancora validi oggi
```

Ricerca per ASN o organizzazione

```
whois AS15169            # Google ASN
whois -h whois.ripe.net 150.146.0.0
```

▷ Il RIR di competenza per l'Italia è **RIPE NCC** (`ripe.net`). Puoi cercare qualsiasi prefisso anche via web su `apps.db.ripe.net`.

curl — IP pubblico

```
curl ifconfig.me             # IP pubblico
curl -s api.ipify.org        # alternativa
curl ipinfo.io               # JSON con geo
curl ipinfo.io/8.8.8.8       # info su IP

# L'IP restituito e' quello del router
# NAT, non del tuo PC -- a meno che
# non ci sia NAT sulla tua connessione.
```

nmap — host discovery

```
# Scopri host attivi nella LAN
nmap -sn 192.168.1.0/24
# -sn = ping scan, no port scan

# Output:
# Nmap scan report for 192.168.1.1
# Host is up (0.002s latency).
# MAC Address: AA:BB:CC:DD:EE:FF (Router)

# Solo indirizzi IP (no MAC, no latenza)
nmap -sn 192.168.1.0/24 \
  -oG - | grep "Up" | awk '{print $2}'
```

[!] Esegui `nmap` solo su reti di cui hai l'autorizzazione. Sui laboratori universitari verifica prima con il docente.

dig / nslookup — DNS

Risoluzione forward (nome -> IP)

```
dig google.com               # record A (IPv4)
dig google.com A             # esplicito
dig google.com AAAA          # IPv6
dig +short google.com        # solo IP
dig @8.8.8.8 google.com      # usa DNS 8.8.8.8

# Risposta:
# google.com.  300  IN  A  142.250.180.46
#              ^^^          ^^^^^^^^^^^^^^
#              TTL (sec)    indirizzo IPv4
```

Reverse lookup (IP -> nome)

```
dig -x 8.8.8.8
# 8.8.8.8.in-addr.arpa --> dns.google.

dig -x 150.146.1.1
# --> hostname UniMe
```

nslookup (portabile, anche Windows)

```
nslookup google.com
nslookup google.com 8.8.8.8  # DNS custom
nslookup -type=MX gmail.com  # record MX
```

▷ In C usa `getaddrinfo()` per la risoluzione DNS: supporta IPv4 e IPv6, è thread-safe, rimpiazza la deprecata `gethostbyname()`.

API C — indirizzi IP

Struttura sockaddr_in

```c
struct sockaddr_in {
    sa_family_t sin_family; /* AF_INET   */
    in_port_t   sin_port;   /* htons(p)  */
    struct in_addr sin_addr;/* inet_pton */
};
/* sin_addr.s_addr: 32bit network order */
```

Testo <-> binario

```c
/* stringa --> struct in_addr */
inet_pton(AF_INET, "192.168.1.47",
          &addr.sin_addr);

/* struct in_addr --> stringa */
char buf[INET_ADDRSTRLEN]; /* 16 byte */
inet_ntop(AF_INET, &addr.sin_addr,
          buf, sizeof(buf));
```

[!] Non usare `inet_ntoa()`: buffer statico interno, non thread-safe, deprecata. Usa sempre `inet_ntop()`.

Lista interfacce locali

```c
#include <ifaddrs.h>
struct ifaddrs *ifa_list, *ifa;
getifaddrs(&ifa_list);
```

```c
for (ifa = ifa_list; ifa; ifa=ifa->ifa_next){
    if (!ifa->ifa_addr) continue;
    if (ifa->ifa_addr->sa_family != AF_INET)
        continue;
    struct sockaddr_in *sa =
        (struct sockaddr_in*)ifa->ifa_addr;
    char buf[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &sa->sin_addr,
              buf, sizeof(buf));
    printf("%-10s %s\n",
           ifa->ifa_name, buf);
}
freeifaddrs(ifa_list);
/* come: ip addr show */
```

Risoluzione DNS in C

```c
#include <netdb.h>
struct addrinfo hints={0}, *res, *p;
hints.ai_family   = AF_INET;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo("google.com","80",&hints,&res);
for (p=res; p; p=p->ai_next) {
    struct sockaddr_in *sa =
        (struct sockaddr_in*)p->ai_addr;
    char buf[INET_ADDRSTRLEN];
    inet_ntop(AF_INET,&sa->sin_addr,
              buf,sizeof(buf));
    printf("-> %s\n", buf);
}
freeaddrinfo(res);
/* come: dig +short google.com */
```

▷ Un hostname può avere più IP: `getaddrinfo()` restituisce una lista. Itera sempre su tutti i risultati.

Riferimento rapido: indirizzi speciali

Blocco	Uso
10.0.0.0/8	Privato RFC 1918
172.16.0.0/12	Privato RFC 1918
192.168.0.0/16	Privato RFC 1918
127.0.0.0/8	Loopback
0.0.0.0	INADDR_ANY (bind)
169.254.0.0/16	Link-local (APIPA)
224.0.0.0/4	Multicast
255.255.255.255	Broadcast locale
::1/128	Loopback IPv6
fe80::/10	Link-local IPv6
fc00::/7	Privato IPv6

Riferimento: gerarchia assegnazione

Ente	Ruolo
IANA	Assegna ai RIR
RIPE NCC	Europa / Medio Oriente
ARIN	Nord America
APNIC	Asia-Pacifico
LACNIC	America Latina
AFRINIC	Africa
ISP / LIR	Assegna agli utenti

▷ **Cronologia**: 1993 CIDR (RFC 1519) → 1994 NAT (RFC 1631) → 1995 IPv6 (RFC 1883) → 1996 indirizzi privati (RFC 1918) → 2019 RIPE NCC esaurito.