# CN Practice - Assignment 1

G S Santhosh Raghul
COE18B045

## Linux Commands

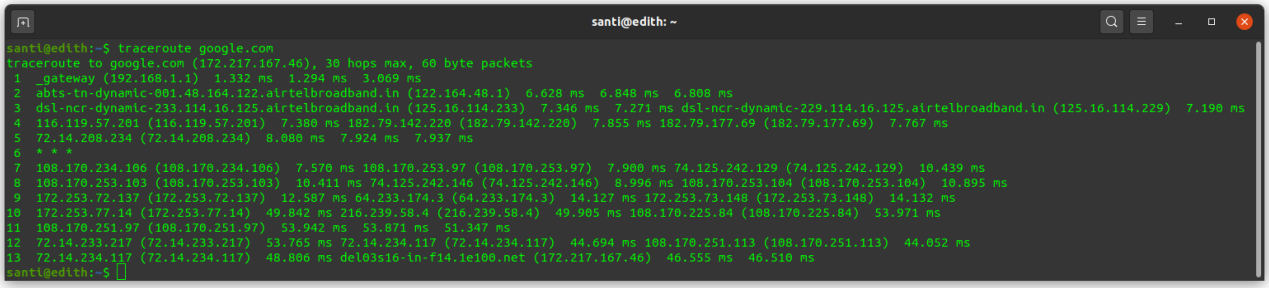
ifconfig:

Displays all the active interfaces details. It is also used to check the ip address.
With the -a argument, it displays information of all active or inactive network interfaces

```
santi@edith: ~
santi@edith:~$ ifconfig -a
eno1: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether ec:8e:b5:56:79:7f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 341959  bytes 48105449 (48.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 341959  bytes 48105449 (48.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.4  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fd1c:6758:72d0:6f00:8e43:a61b:8b2:87fa  prefixlen 64  scopeid 0x0<g
lobal>
        inet6 fd1c:6758:72d0:6f00:bd9e:870b:c159:e195  prefixlen 64  scopeid 0x0<
global>
        inet6 fe80::178f:aa9f:6c25:84f7  prefixlen 64  scopeid 0x20<link>
        ether 30:e3:7a:0e:30:bc  txqueuelen 1000  (Ethernet)
        RX packets 2582430  bytes 1807010778 (1.8 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 821550  bytes 147213734 (147.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

santi@edith:~$
```

traceroute:

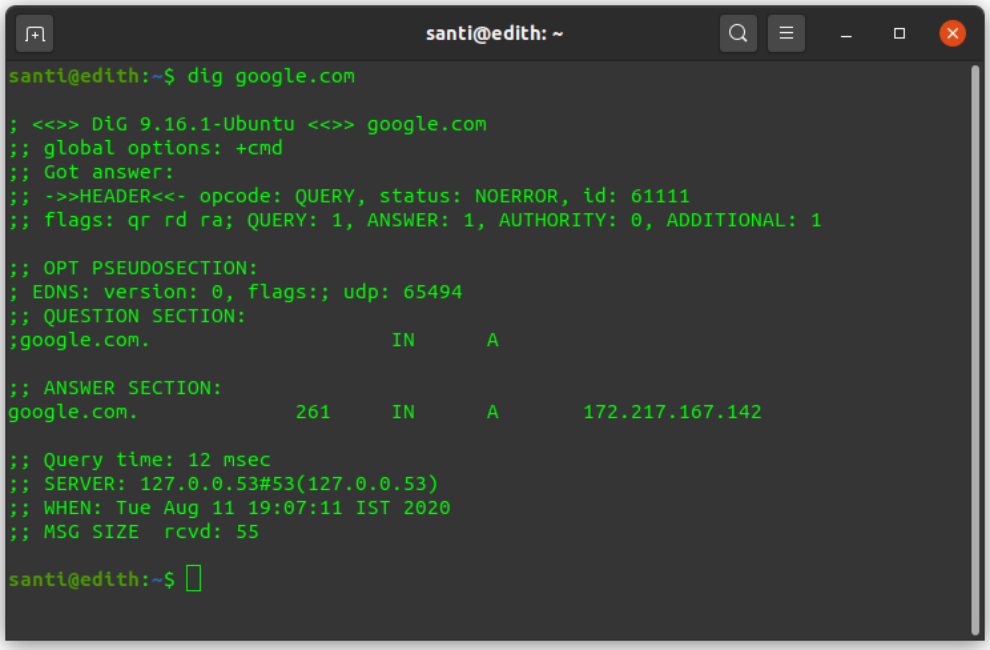
traceroute followed by ip address or domain name prints the route that a packet takes to reach the host.



```
santi@edith: ~
santi@edith:~$ traceroute google.com
traceroute to google.com (172.217.167.46), 30 hops max, 60 byte packets
 1  _gateway (192.168.1.1)  1.332 ms  1.294 ms  3.069 ms
 2  abts-tn-dynamic-001.48.164.122.airtelbroadband.in (122.164.48.1)  6.628 ms  6.848 ms  6.808 ms
 3  dsl-ncr-dynamic-233.114.16.125.airtelbroadband.in (125.16.114.233)  7.346 ms  7.271 ms dsl-ncr-dynamic-229.114.16.125.airtelbroadband.in (125.16.114.229)  7.190 ms
 4  116.119.57.201 (116.119.57.201)  7.380 ms 182.79.142.220 (182.79.142.220)  7.855 ms 182.79.177.69 (182.79.177.69)  7.767 ms
 5  72.14.208.234 (72.14.208.234)  8.080 ms  7.924 ms  7.937 ms
 6  * * *
 7  108.170.234.106 (108.170.234.106)  7.570 ms 108.170.253.97 (108.170.253.97)  7.900 ms 74.125.242.129 (74.125.242.129)  10.439 ms
 8  108.170.253.103 (108.170.253.103)  10.411 ms 74.125.242.146 (74.125.242.146)  8.996 ms 108.170.253.104 (108.170.253.104)  10.895 ms
 9  172.253.72.137 (172.253.72.137)  12.587 ms 64.233.174.3 (64.233.174.3)  14.127 ms 172.253.73.148 (172.253.73.148)  14.132 ms
10  172.253.77.14 (172.253.77.14)  49.842 ms 216.239.58.4 (216.239.58.4)  49.905 ms 108.170.225.84 (108.170.225.84)  53.971 ms
11  108.170.251.97 (108.170.251.97)  53.942 ms  53.871 ms  51.347 ms
12  72.14.233.217 (72.14.233.217)  53.765 ms 72.14.234.117 (72.14.234.117)  44.694 ms 108.170.251.113 (108.170.251.113)  44.052 ms
13  72.14.234.117 (72.14.234.117)  48.806 ms del03s16-in-f14.1e100.net (172.217.167.46)  46.555 ms  46.510 ms
santi@edith:~$
```

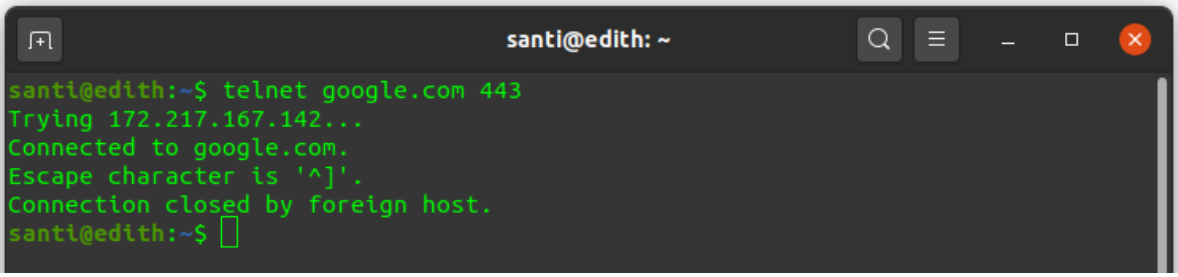
dig:

dig (Domain Information Groper) is used for querying Domain Name System (DNS) name servers. It performs DNS lookups and displays the answers that are returned from the name server that were queried.



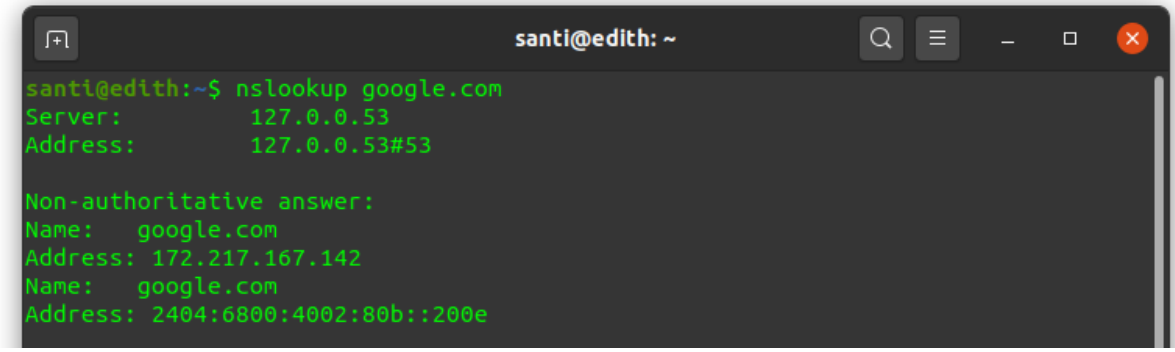

telnet:

connect destination host:port via a telnet protocol if connection establishes means connectivity between two hosts is working fine.



nslookup:

nslookup (Name Server Lookup) is a used for getting information from DNS (Domain Name System) server. It is a network administration tool for querying the DNS to obtain domain name or IP address mapping or any other specific DNS record.




netstat:

netstat displays various network related information such as network connections, routing tables, interface statistics, masquerade connections, multicast memberships etc.

ping:

it stands for (Packet Internet Groper). It checks connectivity between two nodes to see if a server is available. It checks if a remote host is up, or that network interfaces can be reached. It also helps checking your network connection and verifying network issues.Ping command keeps executing and sends the packet until you interrupt.

curl or wget:

used to download a file and save it in the current directory.

Syntax:

Curl -O <link>

wget <link>

nmap:

It checks the opened port on the server. Used for network exploration and security auditing.

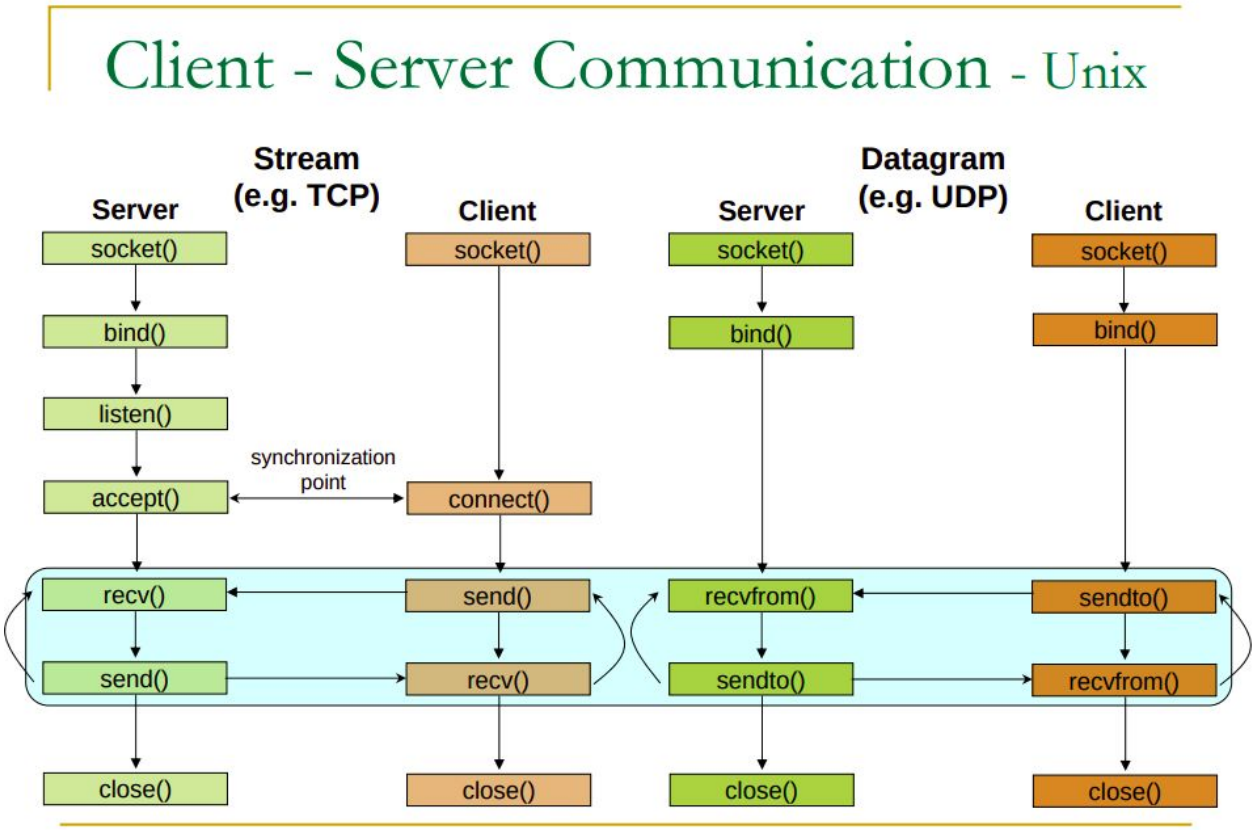
scp:

allows you to secure copy files to and from another host in the network.

Syntax:

scp source_file_name username@destination_host:destination_folder

# Socket API documentation




## socket() :

int sockid = socket(domain, type, protocol)

**domain** : an integer, communication domain
e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
**type** : communication type
SOCK_STREAM for TCP(reliable, connection oriented)
SOCK_DGRAM for UDP(unreliable, connectionless)
**protocol** : it specifies the protocol. Usually set to 0 for Internet Protocol. This is the same **number** which appears on protocol field in the IP header of a packet.

This function **returns** an integer which is a socket descriptor (just like a file-handle) when a socket has been created and returns -1 in case of failure.

## Socket API defines a generic data type for addresses:

```
struct sockaddr
{
unsigned short sa_family; /* Address family (e.g. AF_INET) */
char sa_data[14]; /* Family-specific address information */
}
```

## Particular form of the sockaddr used for TCP/IP addresses:

```
struct in_addr
{
unsigned long s_addr; /* Internet address (32 bits) */
}
struct sockaddr_in
{
unsigned short sin_family; /* Internet protocol (AF_INET) */
unsigned short sin_port; /* Address port (16 bits) */
struct in_addr sin_add; /* Internet address (32 bits) */
char sin_zero[8]; /* Not used */
}
```

sockaddr_in can be type casted to a sockaddr

## bind() :

```
int status = bind(sockid, &addrport, size);
```

**sockid**: integer, socket descriptor
**addrport**: struct sockaddr which contains the (IP) address and port of the machine
for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface)
**size**: the size (in bytes) of the addrport structure
This function **returns** -1 upon failure.

## listen():

```
int status = listen(sockid, queueLimit);
```

**sockid**: integer, socket descriptor
**queueLimit**: integer, number of active participants that can “wait” for a connection
**returns** 0 if listening, -1 if error
The listening socket (sockid) is never used for sending and receiving but only as a way to get new sockets in TCP

## connect() :

The client establishes a connection with the server by calling this function.

```
int status = connect(sockid, &foreignAddr, addrlen);
```

**sockid**: integer, socket to be used in connection
**foreignAddr**: struct sockaddr: address of the passive participant
**addrlen**: integer, sizeof(name)
**returns** 0 if successful connect, -1 otherwise

## accept() :

The server gets a socket for an incoming client connection by calling this function.

```
int s = accept(sockid, &clientAddr, &addrLen);
```

**sockid**: integer, the orig. socket (being listened on)
**clientAddr**: struct sockaddr, address of the active participant
**addrLen**: sizeof(clientAddr): value/result parameter
- must be set appropriately before call
- adjusted upon return

**returns** an integer, the new socket (used for data-transfer)

## Exchanging data with stream socket - send() and recv() :

```
int count = send(sockid, msg, msgLen, flags);
```

**msg**: const void[], message to be transmitted
**msgLen**: integer, length of message (in bytes) to transmit
**flags**: integer, special options, usually just 0
**Returns** # bytes transmitted (-1 if error)

```
int count = recv(sockid, recvBuf, bufLen, flags);
```

**recvBuf**: void[], stores received bytes
**bufLen**: # bytes received
**flags**: integer, special options, usually just 0
**returns** # bytes received (-1 if error)

## Exchanging data with dgram socket - sendto() and recvfrom() :

```
int count = sendto(sockid, msg, msgLen, flags,&foreignAddr, addrlen);
```

**msg**, **msgLen**, **flags**, **return value**: same with send()
**foreignAddr**: struct sockaddr, address of the destination
**addrLen**: sizeof(foreignAddr)

```
int count = recvfrom(sockid, recvBuf, bufLen,flags, &clientAddr, addrlen);
```

**recvBuf**, **bufLen**, **flags**, **return value**: same with recv()
**clientAddr**: struct sockaddr, address of the client
**addrLen**: sizeof(clientAddr)

## close() :

When finished using a socket, the socket should be closed.

```
status = close(sockid);
```

**sockid**: the socket descriptor
**Returns** 0 if successful, -1 if error
Closing a socket closes a connection (for stream socket) and frees up the port used by the socket.

# UDP chat program

My program uses udp to transfer data the client and the server. Both the client and the server program terminate when any one of client or server sends a “bye” message.