**Ip**

KEY TAKEAWAYS

The ip command has replaced the older ifconfig command in modern versions of Linux.

The ip command allows you to configure IP addresses, network interfaces, and routing rules on the fly without rebooting.

Run "ip addr" in the Terminal to get your PC's local IP address.

**how you can use this modern replacement of the classic (and now deprecated) ifconfig .?**

**How the ip Command Works**

*With the ip command, you can*[*adjust the way a Linux computer*](http://man7.org/linux/man-pages/man8/ip.8.html)*handles IP addresses,*[*network interfaces controllers*](https://en.wikipedia.org/wiki/Network_interface_controller)*(NICs), and*[*routing rules*](https://en.wikipedia.org/wiki/IP_routing)*. The changes also take immediate effect — you don't have to reboot. The ip command can do a lot more than this.*

*The ip command has many subcommands, each of which works on a type of object, such as IP addresses and routes. There are, in turn, many options for each of these objects. It's this richness of functionality that gives the ip command the granularity you need to perform what can be delicate tasks*

**We'll look at the following objects:**

* **Address: IP addresses and ranges.**
* **Link: Network interfaces, such as wired connections and Wi-Fi adapters.**
* **Route: The rules that manage the routing of traffic sent to addresses via interfaces (links ).**

**Using ip with Addresses**

**Obviously, you first have to know the settings you're dealing with. To discover which IP addresses your computer has, you use the ip command with the object “address”. The default action is “show”, which lists the IP addresses. You can also omit “show” and abbreviate “address” as "addr" or even "a."**

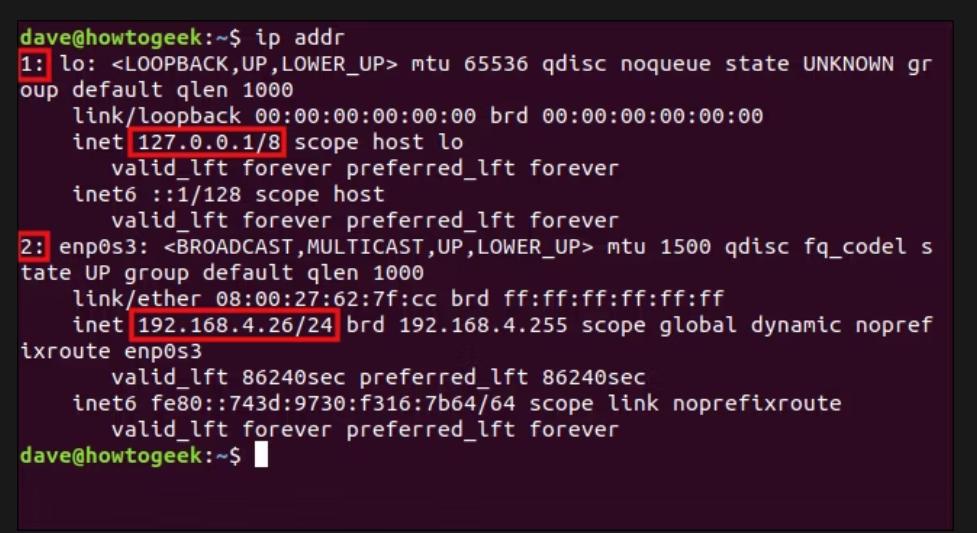
**The following commands are all equivalent:**

**ip address show**

**ip addr show**

**ip addr**

**ip a**

****We see two IP addresses, along with a lot of other information. IP addresses are associated with network interface controllers (NICs). The ip command tries to be helpful and provides a bunch of information about the interface, too.

The first IP address is the (internal) loopback address used to communicate within the computer. The second is the actual (external) IP address the computer has on the local area network (LAN).

Let's break down all the information we received:

* **lo**: The network interface name as a string.
* **<LOOPBACK,UP,LOWER\_UP>:** This is a loopback interface. It's UP, meaning it's operational. The [physical networking layer](https://en.wikipedia.org/wiki/OSI_model) (layer one) is also up.
* **mtu 65536:** The maximum transfer unit. This is the size of the largest chunk of data this interface can transmit.
* **qdisc noqueue:** A qdisc is a queuing mechanism. It schedules the transmission of packets. There are different queuing techniques called disciplines. The noqueue discipline means "send instantly, don't queue." This is the default qdisc discipline for virtual devices, such as the loopback address.
* **state UNKNOWN:** This can be DOWN (the network interface is not operational), UNKNOWN (the network interface is operational but nothing is connected), or UP (the network is operational and there is a connection).
* **group default:** Interfaces can be grouped logically. The default is to place them all in a group called "default."
* **qlen 1000:** The maximum length of the transmission queue.
* **link/loopback:** The [media access control](https://en.wikipedia.org/wiki/MAC_address) (MAC) address of the interface.
* **inet 127.0.0.1/8:** The IP version 4 address. The part of the address after the forward-slash (/) is [Classless Inter-Domain Routing notation](https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing#CIDR_notation) (CIDR) representing the subnet mask. It indicates how many leading contiguous bits are set to one in the subnet mask. The value of eight means eight bits. Eight bits set to one represents 255 in binary, so the subnet mask is 255.0.0.0.
* **scope host:** The IP address scope. This IP address is only valid inside the computer (the "host").
* **lo:** The interface with which this IP address is associated.
* **valid\_lft:** Valid lifetime. For an IP version 4 IP address allocated by [Dynamic Host Configuration Protocol](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol) (DHCP), this is the length of time the IP address is considered valid and able to make and accept connection requests.
* **preferred\_lft:** Preferred lifetime. For an IP version 4 IP address allocated by DHCP, this is the amount of time the IP address can be used with no restrictions. This should never be larger than the valid\_lft value.
* **inet6**: The IP version 6 address, scope , valid\_lft, and preferred\_lft.

The physical interface is more interesting, as we'll show below:

* **enp0s3:** The network interface name as a string. The "en" stands for ethernet, "p0" is the bus number of the ethernet card, and "s3" is the slot number.
* **<BROADCAST,MULTICAST,UP,LOWER\_UP>:** This interface supports [broad-](https://en.wikipedia.org/wiki/Broadcasting_(networking)) and [multicasting](https://en.wikipedia.org/wiki/Multicast), and the interface is UP (operational and connected). The hardware layer of the network (layer one) is also UP.
* **mtu 1500:** The maximum transfer unit this interface supports.
* **qdisc fq\_codel:** The scheduler is using a discipline called "Fair Queuing, Controlled Delay." It's designed to provide a fair share of the bandwidth to all the traffic flows that use the queue.
* **state UP:** The interface is operational and connected.
* **group default:** This interface is in the "default" interface group.
* **qlen 1000:** The maximum length of the transmission queue.
* **link/ether:** The MAC address of the interface.
* **inet 192.168.4.26/24:** The IP version 4 address. The "/24" tells us there are 24 contiguous leading bits set to one in the subnet mask. That's three groups of eight bits. An eight-bit binary number equates to 255; therefore, the subnet mask is 255.255.255.0.
* **brd 192.168.4.255:** The [broadcast address](https://en.wikipedia.org/wiki/Broadcast_address) for this subnet.
* **scope global:** The IP address is valid everywhere on this network.
* **dynamic:** The IP address is lost when the interface goes down.
* **noprefixroute:** Do not create a route in the route table when this IP address is added. Someone has to add a route manually if he wants to use one with this IP address. Likewise, if this IP address is deleted, don't look for a route to delete.
* **enp0s3:** The interface with which this IP address is associated.
* **valid\_lft:** Valid lifetime. The time the IP address will be considered valid; 86,240 seconds is 23 hours and 57 minutes.
* **preferred\_lft:** Preferred lifetime. The time the IP address will operate without any restrictions.
* **inet6:** The IP version 6 address, scope, valid\_lft, and preferred\_lft.

**Display Only IPv4 or IPv6 Addresses**

If you want to limit the output to the IP version 4 addresses, you can use the -4 option, as follows:

ip -4 addr

If you want to limit the output to the IP version 6 addresses, you can use the -6 option, as follows:

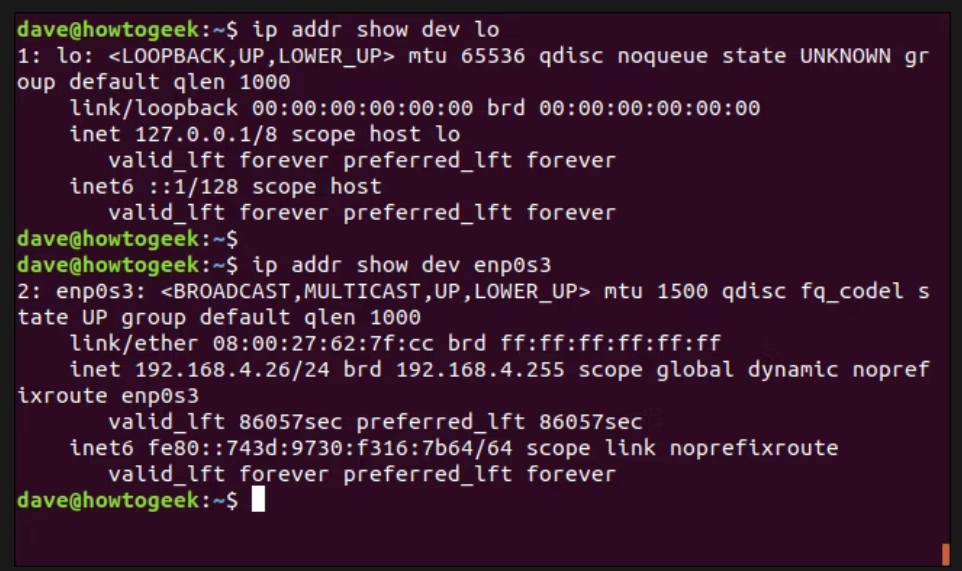
ip -6 addr

**Display Information for a Single Interface**

If you want to see the IP address information for a single interface, you can use the show and dev options, and name the interface, as shown below:

ip addr show dev lo

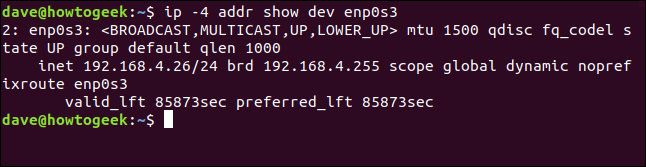
ip addr show dev enp0s3



You can also use the -4 or -6 flag to further refine the output so you only see that in which you're interested.

If you want to see the IP version 4 information related to the addresses on interface enp0s3, type the following command:

ip -4 addr show dev enp0s3



For study more functionality of ip refer to:-

<https://www.howtogeek.com/657911/how-to-use-the-ip-command-on-linux/>

**dig(Domain Information Groper)**

*The Linux dig command allows you to query DNS servers and perform DNS lookups. You can also find the domain an IP address leads back to.*

## How the dig Command Works

People use the Linux dig command to query[Domain Name System (DNS)](https://www.howtogeek.com/122845/htg-explains-what-is-dns/) servers. dig is an acronym for [Domain Information Groper](https://linux.die.net/man/1/dig). With dig, you can query DNS servers for information regarding various DNS records, including host addresses, mail exchanges, name servers, and related information. It was intended to be a tool for diagnosing DNS issues. However, you can use it to poke around and learn more about DNS, which is one of the central systems that keep the internet routing traffic.

The internet uses [internet protocol (IP) addresses](https://www.howtogeek.com/341307/how-do-ip-addresses-work/) to identify "locations" around the web, but people use domain names. When you type a domain name into an application, like a web browser or [SSH client](https://www.howtogeek.com/311287/how-to-connect-to-an-ssh-server-from-windows-macos-or-linux/), something has to translate from the domain name to the actual IP address. This is where the Domain Name System comes in.

When you use a domain name with any internet-connected program, your local router can't resolve it (unless it's cached from a previous request). So, your router queries either your Internet Service Provider's (ISP) DNS server, or any other you've configured your system to use. These are called DNS precursor servers.

If the DNS server recently received the same request from someone else on the same computer, the answer might be in its cache. If that's the case, it simply sends that same information back to your program.

If the DNS precursor server can't locate the domain in its cache, it contacts a DNS [root name server](https://en.wikipedia.org/wiki/Root_name_server). A root server won't hold the information required to resolve domain names to IP addresses, but it will hold lists of servers that can help with your request.

The root server looks at the [top-level domain](https://en.wikipedia.org/wiki/Top-level_domain) to which your domain name belongs, such as .COM, .ORG, .CO.UK, and so on. It then sends a list of the top-level domain servers that handle those types of domains back to the DNS precursor server. The DNS precursor server can then make its request once more, to a top-level domain server.

The top-level domain server sends the details of the [authoritative name server](https://en.wikipedia.org/wiki/Name_server#Authoritative_name_server) (where the details of the domain are stored) back to the DNS precursor server. The DNS server then queries the authoritative name server that's hosting the zone of the domain you originally entered into your program. The authoritative name server sends the IP address back to the DNS server, which, in turn, sends it back to you.

## Installing dig

dig was already installed on our Ubuntu 18.04 and Fedora 30 computers. However, we had to install it on the Manjaro 18.04 computer with the following command:

sudo pacman -Sy bind-tools

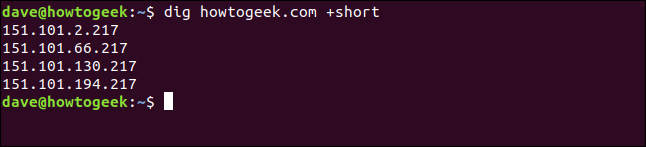
sudo pacman -Sy bind-tools in a terminal window

## Getting Started with dig

In our first example, we'll return the IP addresses associated with a domain name. Often, multiple IP addresses are associated with a single domain name. This often happens if load balancing is used, for example.

We use the +short query option, as shown below, which gives us a terse response:

dig howtogeek.com +short



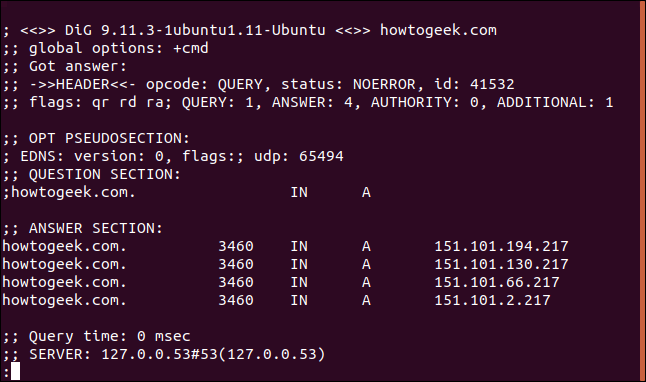
All the IP addresses associated with the howtogeek.com domain are listed for us. At the other end of the spectrum, if we don't use the +short query option, the output is quite verbose.

So, we type the following to pipe it through less:

dig howtogeek.com | less

dig howtogeek.com | less in a terminal window

The output is displayed in less, as shown below.



Here's the full listing:

; <<>> **DiG** 9.11.3-1ubuntu1.11-Ubuntu <<>> **howtogeek**.com  
;; **global** options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12017  
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1  
;; OPT PSEUDOSECTION:  
; EDNS: version: 0, flags:; udp: 65494  
;; QUESTION SECTION:  
;**howtogeek**.com. **IN** **A**  
;; ANSWER SECTION:  
**howtogeek**.com. 3551 **IN** **A** 151.101.194.217  
**howtogeek**.com. 3551 **IN** **A** 151.101.130.217  
**howtogeek**.com. 3551 **IN** **A** 151.101.66.217  
**howtogeek**.com. 3551 **IN** **A** 151.101.2.217  
;; Query time: 0 msec  
;; **SERVER**: 127.0.0.53#53(127.0.0.53)  
;; **WHEN**: **Sun** **Mar** 22 07:44:37 **EDT** 2020  
;; MSG SIZE rcvd: 106

Let's dissect that piece by piece.

### Header

First, let's take a look at we have in the Header:

; <<>> **DiG** 9.11.3-1ubuntu1.11-Ubuntu <<>> **howtogeek**.com  
;; **global** options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12017  
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

Now, here's what all of that means:

* **First line:** The version of dig and the domain that was queried.
* **Global options:**As we'll see, you can use dig to query multiple domains simultaneously. This line shows the options that have been applied to all of the domain queries. In our simple example, it was just the default +cmd (command) option.
* **Opcode: Query:** This is the type of operation that was requested which, in this case, was a query. This value can also be iquery for an inverse query, or status if you're just testing the state of the DNS system.
* **Status: Noerror:** There were no errors and the request was correctly resolved.
* **ID: 12017**: This random ID ties the request and response together.
* **Flags: qr rd ra:** These stand for query, recursion desired, and recursion available. Recursion is one form of DNS lookup (the other is iterative). You might also see AA, which stands for Authoritative Answer, meaning an Authoritative Name Server provided the response.
* **Query: 1:** The number of queries in this session, which was one.
* **Answer: 4:** The number of answers in this response, which is four.
* **Authority: 0:** The number of answers that came from an Authoritative Name Server, which was zero in this case. The response was returned from the cache of a DNS precursor server. There will be no authoritative section in the response.
* **Additional: 1:** There is one piece of additional information. (Strangely, nothing is listed unless this value is two or higher.)

### ****Opt Pseudosection****

Next, we see the following in the Opt Pseudosection:

;; OPT PSEUDOSECTION:  
; EDNS: version: 0, flags:; udp: 65494

Let's break that down:

* **EDNS: version 0:** The version of [Extension System for DNS](https://en.wikipedia.org/wiki/Extension_mechanisms_for_DNS) that's being used. EDNS transmits extended data and flags by extending the size of the [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP) packets. This is indicated by a variable size flag.
* **flags:** No flags are in use.
* **udp**: **4096:** The UDP packet size.

### ****Question Section****

In the Question section, we see the following:

;; QUESTION SECTION:  
;**howtogeek**.com. **IN** **A**

Here's what this means:

* **howtogeek.com:** The domain name we're querying.
* **IN:** We're making an internet class query.
* **A:** Unless we specify otherwise, dig will request an A (address) record from the DNS server.

**Answer Section**

The Answer section contains the following four answers we received from the DNS server:

**howtogeek**.com. 3551 **IN** **A** 151.101.194.217  
**howtogeek**.com. 3551 **IN** **A** 151.101.130.217  
**howtogeek**.com. 3551 **IN** **A** 151.101.66.217  
**howtogeek**.com. 3551 **IN** **A** 151.101.2.217

Here's what these answers mean:

* **3551:** This is the Time to Live (TTL), a 32-bit signed integer that holds the time interval for which a record can be cached. When it expires, the data must be used in an answer to a request until it's been refreshed by the DNS server.
* **IN:** We made an Internet class query.
* **A:** We asked for an A record from the DNS server.

### ****Statistics Section****

Statistics is the final section, and it contains the following information:

;; Query time: 0 msec   
;; **SERVER**: 127.0.0.53#53(127.0.0.53)   
;; **WHEN**: **Sun** **Mar** 22 07:44:37 **EDT** 2020   
;; MSG SIZE rcvd: 106

Here's what we've got:

* **Query Time: 0 msec:** The time it took to get the response.
* **SERVER: 127.0.0.53#53(127.0.0.53):** The IP Address and port number of the DNS server that responded. In this case, it's pointing to the local caching stub resolver. This forwards DNS requests to whichever upstream DNS servers are configured. On the Manajro test computer, the address listed here was 8.8.8.8#53, which is [Google's public DNS service](https://en.wikipedia.org/wiki/Google_Public_DNS).
* **WHEN: Sun Mar 22 07:44:37 EDT 2020:** When the request was made.
* **MSG SIZE rcvd: 106:** The size of the message received from the DNS server.

## Being Selective

You don't have to settle for the two extremes of tight-lipped and garrulous. The dig command allows you to selectively include or exclude sections from the results.

The following query options will remove that section from the results:

* **+nocomments:** Don't show comment lines.
* **+noauthority:** Don't show the authority section.
* **+noadditional:** Don't show the additional section.
* **+nostats:** Don't show the stats section.
* **+noanswer:** Don't show the answer section.
* **+noall:** Don't show anything!

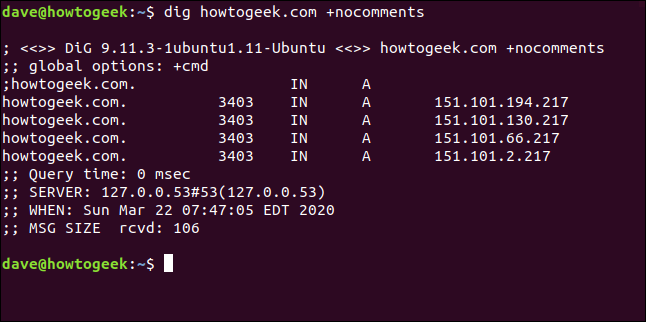
The +noall query option is usually combined with one of those above to include a section in the results. So, instead of typing a long string of query options to turn off multiple sections, you can use +noall to turn them all off.

You can then use the following inclusive query options to turn those you want to see back on:

* **+comments:** Show comment lines.
* **+authority:** Show the authority section.
* **+additional:** Show the additional section.
* **+stats:** Show the stats section.
* **+answer:** Show the answer section.
* **+all:** Show everything.

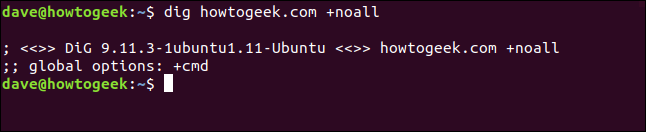
We type the following to make a request and exclude the comment lines:

dig howtogeek.com +nocomments



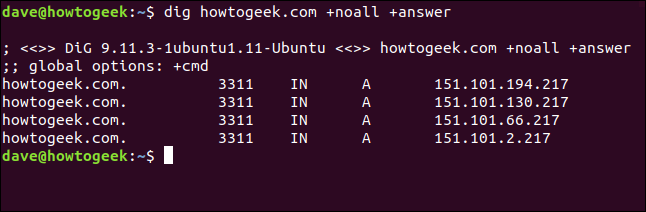
If we use the +noall query option on its own, as shown below, we won't get any useful output:

dig howtogeek.com +noall



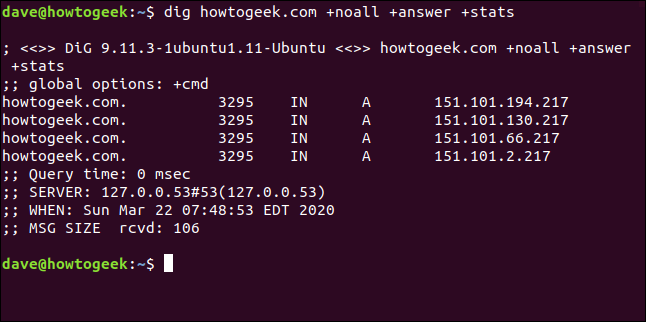
We can selectively add the sections we want to see. To add the answer section, we type the following:

dig howtogeek.com +noall +answer



If we type the following to turn on +stats, we'll also see the statistics section:

dig howtogeek.com +noall +answer +stats



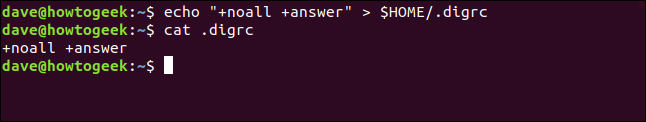
The +noall +answer combination is used often. You can add other sections to the command line as required. If you want to avoid typing +noall +answer on the command line every time you use dig, you can put them in a configuration file called ".digrc." It's located in your home directory.

We type the following to create one [with echo](https://www.howtogeek.com/446071/how-to-use-the-echo-command-on-linux/):

echo "+noall +answer" > $HOME/.digrc

We can then type the following to check its contents:

cat .digrc

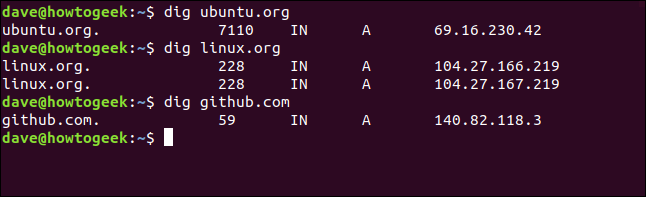


Those two options will now be applied to all future uses of dig, as shown below:

dig ubuntu.org

dig linux.org

dig github.com



This dig configuration file will be in use for the remaining examples in this article.

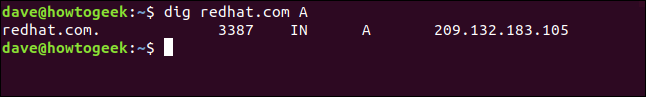
## DNS Records

The information returned to your dig requests is pulled from different types of records held on the DNS server. Unless we ask for something different, dig queries the A (address) record. The following are the types of records commonly used with dig:

* **A Record:** Links the domain to an IP version 4 address.
* **MX Record:** Mail exchange records direct emails sent to domains to the correct mail server.
* **NS Record:** Name server records delegate a domain (or subdomain) to a set of DNS servers.
* **TXT Record:** Text records store text-based information regarding the domain. Typically, they might be used to suppress spoofed or forged email.
* **SOA Record:** Start of authority records can hold a lot of information about the domain. Here, you can find the primary name server, the responsible party, a timestamp for changes, the frequency of zone refreshes, and a series of time limits for retries and abandons.
* **TTL:** Time to live is a setting for each DNS record that specifies how long a DNS precursor server is allowed to cache each DNS query. When that time expires, the data must be refreshed for subsequent requests.
* **ANY:** This tells dig to return every type of DNS record it can.

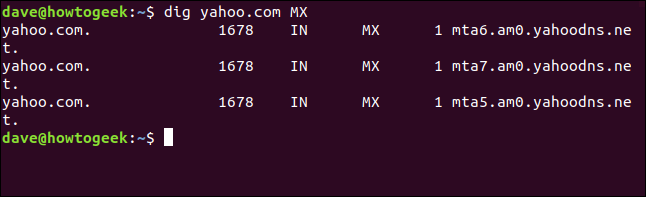
Specifying the A record type doesn't change the default action, which is to query the address record and obtain the IP address, as shown below:

dig redhat.com A



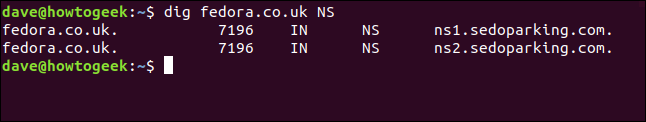
To query the mail exchange records, we use the following MX flag:

dig yahoo.com MX



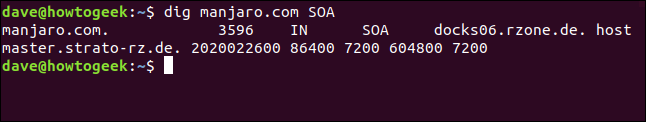
The name server flag returns the following name of the root name servers associated with the top-level domain:

dig fedora.com NS



To query the start of authority record, we type the following SOA flag:

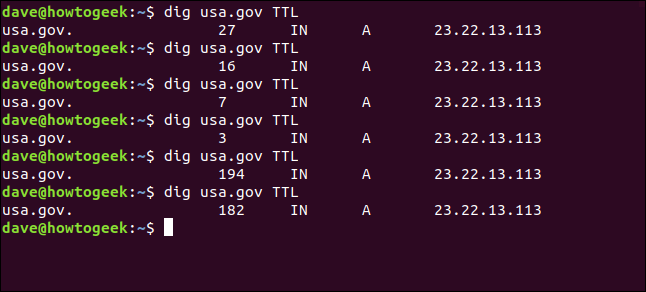
dig manjaro.com SOA



The TTL flag will show us the time to live for the data in the DNS server's cache. If we make a series of requests, we see the time to live reduce to nothing, and then jump back to its starting value.

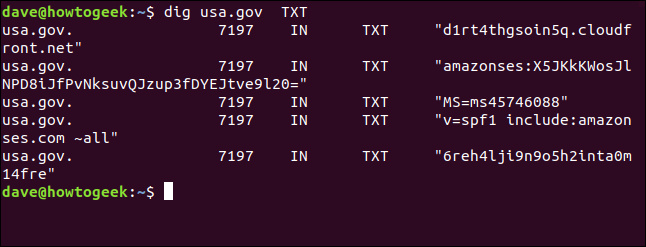
We type the following:

dig usa.gov TTL



To see the text records, we type the TX flag:

dig usa.gov TXT



## Specifying the DNS Server

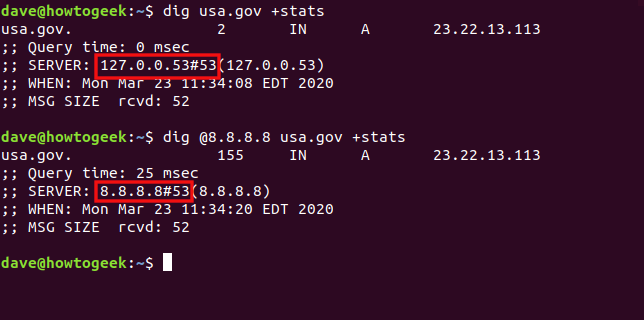
If you want to use a particular DNS server for your request, you can use the at sign (@) to pass it to dig as a command-line parameter.

With the default DNS server (see below), dig references the local caching stub resolver at 127.0.0.53.

dig usa.gov +stats

Now, we type the following to use Google's public DNS server at 8.8.8.8:

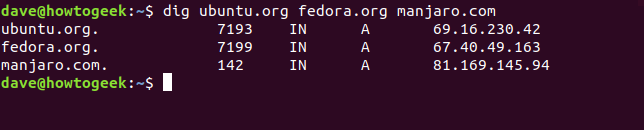
dig @8.8.8.8 usa.gov +stats



## Using dig with Multiple Domains

We can pass multiple domains to dig on the command line, as shown below:

dig ubuntu.org fedora.org manjaro.com

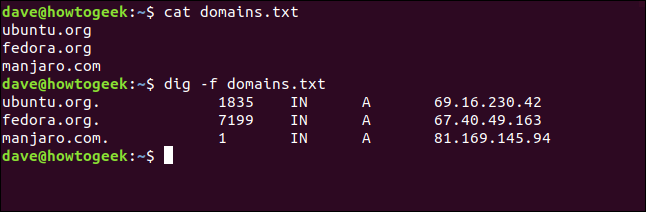


If you regularly check a set of domains, you can store them in a text file and pass it to dig. All the domains in the file will be checked in turn.

Our file is called "domains.txt." We'll use cat to show its contents, and then pass it to dig with the -f (file) option. We type the following:

cat domains.txt

dig -f domains.txt



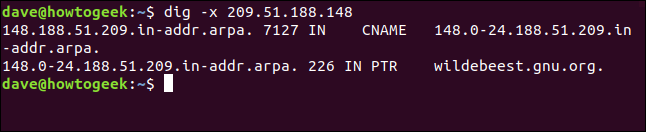
## Reverse DNS Lookups

If you have an IP address and want to know where it goes, you can try a reverse DNS lookup. If it resolves to a server registered with a DNS server, you might be able to find out its domain.

Whether you can depends on the presence of a PTR (pointer record). PTRs resolve an IP address to a [fully qualified domain name](https://en.wikipedia.org/wiki/Fully_qualified_domain_name). However, because these aren't mandatory, they're not always present on a domain.

Let's see if we can find out where the IP address 209.51.188.148 takes us. We type the following, using the -x (reverse lookup) option:

dig -x 209.51.188.148

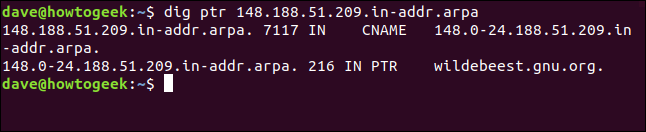


Presto! The IP address resolves to gnu.org.

Because a PTR is a DNS record, and we know dig can request specified DNS records, couldn't we just ask dig to retrieve the PTR for us? Yes, we can, but it does take a bit more work.

We have to provide the IP address in reverse order and tack .in-addr.arpa on the end, as shown below:

dig ptr 148.188.51.209.in-addr.arpa



We get the same result; it just took a bit more effort.

**netstat**

* *The netstat command provides information about network connections, ports in use, and the processes using them.*
* *Netsat can take arguments to filter out unneeded information and only show the specific details you're looking for.*
* *Run sudo netstat -i to list your network interfaces.*

***Ports, Processes, and Protocols***

[*Network sockets*](https://en.wikipedia.org/wiki/Network_socket)*can either be connected or waiting for a connection. The connections use networking protocols like*[*Transport Control Protocol*](https://en.wikipedia.org/wiki/Transmission_Control_Protocol)*(TCP) or*[*User Datagram Protocol*](https://en.wikipedia.org/wiki/User_Datagram_Protocol)*UDP. They use*[*Internet Protocol*](https://en.wikipedia.org/wiki/IPv4)*addresses and*[*network ports*](https://en.wikipedia.org/wiki/Port_(computer_networking))*to establish connections.*

*The word sockets might conjure up images of a physical connection point for a lead or cable, but in this context, a socket is a software construct used to handle one end of a network data connection.*

*Sockets have two main states: They are either connected and facilitating an ongoing network communication, or they are waiting for an incoming connection to connect to them. There are other states, such as the state when a socket is midway through establishing a connection on a remote device, but putting transient states aside, you can think of a socket as either being connected or waiting (which is often called listening).*

*The listening socket is called the server, and the socket that requests a connection with the listening socket is called a client. These names have nothing to do with hardware or computer roles. They simply define the role of each socket at each end of the connection.*

*The netstat command lets you discover which sockets are connected and which sockets are listening. Meaning, it tells you which ports are in use and which processes are using them. It can show you routing tables and statistics about your*[*network interfaces*](https://en.wikipedia.org/wiki/Network_interface_controller)*and*[*multicast connections*](https://en.wikipedia.org/wiki/Multicast)*.*

*The functionality of netstat has been replicated over time in different Linux utilities, such as [ip](http://man7.org/linux/man-pages/man8/ip.8.html) and*[*ss*](http://man7.org/linux/man-pages/man8/ss.8.html)*. It's still worth knowing this granddaddy of all network analysis commands, because it is available on all Linux and Unix-like operating systems, and even on Windows and Mac.*

*Here's how to use it, complete with example commands.*

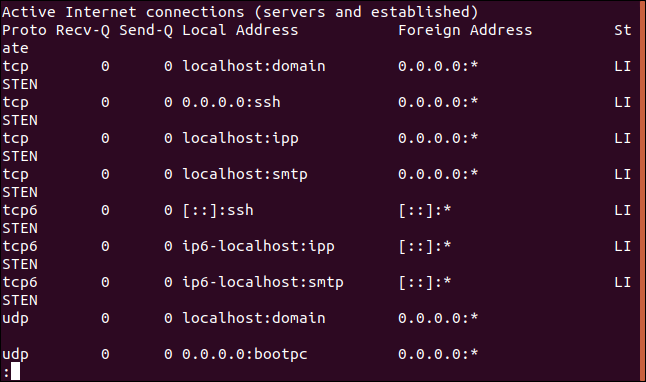
***Listing All Sockets with netstat***

*The -a (all) option makes netstat show all the connected and waiting sockets. This command is liable to produce a long listing, so we pipe it into less.*

*netstat -a | less*

*netstat -a | less in a terminal window*

*The listing includes TCP (IP),*[*TCP6*](https://en.wikipedia.org/wiki/IPv6)*(IPv6), and UDP sockets.*

**

*The wrap-around in the terminal window makes it a little difficult to see what is going on. Here's a couple of sections from that listing:*

*nslookup*

*nslookup is a command-line tool used in various operating systems to query Domain Name System (DNS) servers to obtain domain name or IP address mapping, or other DNS records like MX records (mail exchange), SOA records (start of authority), etc.*

*To use nslookup, you typically open a command prompt or terminal and type nslookup followed by the domain name or IP address you want to look up. For example:*

*nslookup example.com*

*This command will return the IP address associated with the domain name "example.com" along with other DNS information like the authoritative DNS server for that domain.*

*You can also perform reverse DNS lookups by entering an IP address:*

*nslookup 8.8.8.8*

*This command will return the domain name associated with the IP address "8.8.8.8".*

*nslookup can be a useful troubleshooting tool for network connectivity issues or for verifying DNS configurations.*

**nmcli**

*nmcli stands for Network Manager Command-Line Interface. It is a command-line utility for controlling NetworkManager, which is a program that provides network configuration and management for Linux-based systems. nmcli allows users to manage and configure network connections, view connection status, and perform various networking tasks from the command line.*

*Here are some common tasks you can perform with nmcli:*

*Displaying Network Connection Status: You can view the status of all network connections or a specific connection using the following command:*

*nmcli connection show*

**route**

*The* ***route*** *command in Linux is used to view and manipulate the IP routing table. It shows the IP routing table, which contains information about how packets should be forwarded to their destinations. Here are some common uses of the* ***route*** *command:*

1. ***Viewing the Routing Table****: To view the current routing table, you can simply type:*

*route -n*

*The* ***-n*** *option is used to display numeric IP addresses instead of resolving hostnames, which can make the output faster.*

1. ***Adding a Route****: You can add a new route to the routing table using the* ***add*** *option followed by the destination network and gateway:*

*sudo route add -net <destination-network> gw <gateway-ip>*

**Storage management:-**

**MBR**

The Master Boot Record (MBR) is a special type of boot sector at the very beginning of a storage device, such as a hard disk drive or solid-state drive. It plays a crucial role in the boot process of most x86-based computers, including those running Windows, Linux, and other operating systems.

The MBR contains several important components:

1. **Bootstrap Code**: The first 446 bytes of the MBR are reserved for bootstrap code, which is responsible for loading and executing the boot loader. This code is typically specific to the operating system or boot manager installed on the system.
2. **Partition Table**: The next 64 bytes of the MBR contain the partition table, which describes how the disk is divided into partitions. The partition table can contain up to four entries, each describing the starting and ending sectors of a partition, as well as its type and other attributes.
3. **Disk Signature**: The final two bytes of the MBR (at offset 0x1B8) contain a disk signature, which is a unique identifier for the disk. This signature is used by some operating systems and utilities to identify individual disks.

When a computer starts up, the BIOS or UEFI firmware reads the MBR from the boot device into memory and executes the bootstrap code. This code then loads and executes the boot loader, which is responsible for loading the operating system kernel or boot manager.

**ext3 file system**

The ext3 file system is a journaled file system that is commonly used with Linux operating systems. It is an extension of the earlier ext2 file system and was designed to address some of its limitations, particularly regarding data integrity and recovery after system crashes or power failures.

Here are some key features of the ext3 file system:

1. **Journaling**: One of the main features of ext3 is its support for journaling. Journaling helps ensure the consistency of the file system by maintaining a log, or journal, of transactions before they are actually committed to the file system. This helps prevent data corruption and makes recovery faster in the event of a crash.
2. **Backward Compatibility**: Ext3 is designed to be fully backward compatible with ext2, meaning that ext2 file systems can be upgraded to ext3 without losing data. This compatibility extends to utilities and tools, making the transition between the two file systems seamless.
3. **File System Check (fsck) Improvements**: Ext3 includes improvements to the file system check (fsck) utility, which is used to repair inconsistencies in the file system after an unclean shutdown. The journaling feature reduces the need for extensive file system checks and can significantly speed up the recovery process.
4. **Performance**: While ext3's journaling adds some overhead compared to ext2, it generally offers good performance for most workloads. However, it may not be as optimized for certain use cases as other file systems like ext4 or XFS.
5. **Maximum File Size and Partition Size**: Ext3 supports maximum file sizes of up to 16 terabytes and maximum partition sizes of up to 2 terabytes when using 4 KB block sizes. These limitations are higher than those of ext2 but have been surpassed by more modern file systems like ext4 and XFS.

Overall, ext3 strikes a balance between data integrity, backward compatibility, and performance, making it a popular choice for Linux users and administrators, particularly for systems where robustness and stability are priorities. However, it's worth noting that ext4 has largely supplanted ext3 as the default file system for modern Linux distributions due to its improved features and performance.

**Network File System (NFS)**

Network File System (NFS) is a distributed file system protocol that allows a user on a client computer to access files over a network as if they were local files. NFS enables file sharing between computers in a networked environment, providing a simple and efficient way to access files and directories across different platforms.

Here are some key features of NFS:

1. **Client-Server Architecture**: NFS operates on a client-server architecture, where one or more servers host shared directories, known as NFS exports, and clients access these directories over the network. The server makes the file systems available to clients by exporting them, and clients mount these exported file systems as if they were local.
2. **Transparent Access**: NFS provides transparent access to files and directories, allowing clients to read, write, and execute files as if they were stored locally. This transparency makes it easy for users and applications to access remote files without needing to be aware of the underlying network protocols.
3. **Stateless Protocol**: NFS is a stateless protocol, meaning that the server does not keep track of the state of each client. This design simplifies implementation and improves scalability but may require clients to handle certain aspects of file locking and caching to ensure data consistency.
4. **File Locking**: NFS supports file locking mechanisms to prevent multiple clients from simultaneously modifying the same file. This helps maintain data integrity and prevents conflicts when multiple users access shared files concurrently.
5. **Caching**: NFS clients can cache file data and attributes to reduce network overhead and improve performance. However, caching introduces the risk of data inconsistency if multiple clients access the same file simultaneously. Administrators can configure caching behavior based on performance and data consistency requirements.
6. **Security**: NFS provides security features such as authentication and access control to protect shared resources from unauthorized access. These include options for restricting access based on client IP addresses, user permissions, and Kerberos-based authentication for secure communication between clients and servers.

Overall, NFS is widely used in networked environments, including data centers, corporate networks, and cloud computing platforms, to facilitate file sharing and collaboration among users and applications. It offers flexibility, scalability, and interoperability across different operating systems and hardware platforms, making it a popular choice for distributed file storage and access.

**Samba**  
Samba is a software suite that enables file and print services to SMB/CIFS clients (such as Windows-based systems) in a Unix-based environment. SMB (Server Message Block) is a network file sharing protocol, and CIFS (Common Internet File System) is an enhanced version of SMB that offers additional features and improvements.

**NTFS**

The New Technology File System (NTFS) is a proprietary file system developed by Microsoft for use in Windows operating systems. It was introduced with Windows NT 3.1 and has since become the default file system for all versions of Windows, including Windows XP, Windows Vista, Windows 7, Windows 8, and Windows 10. NTFS offers several advanced features and improvements over previous file systems used in Windows, such as FAT (File Allocation Table) and FAT32.

**Cloud and Virtualization:-**

1. OVF and OVA Templates

OVF (Open Virtualization Format) and OVA (Open Virtual Appliance) are both standards for packaging and distributing virtual appliances or virtual machine (VM) templates. These formats are commonly used in virtualization environments, such as VMware vSphere, VirtualBox, and other virtualization platforms.

1. Container Technology and Docker Basics
2. Types of Cloud
3. Cloud Concepts
4. Network Address Translation(NAT)

**Software Management:-**

1. Red Hat Package Manager(RPM)
2. Advanced Package Tool(APT)
3. tar, tgz and gz packages
4. curl and wget

**User and Group management:-**

1. Commands- useradd, groupadd, usermod. groupmod, userdel, groupdel, passwd, change, id, whoami, who, w, last
2. /etc/passwd, /etc/shadow, and /etc/group files

**Service Management:-**

1. systemd
2. systemctl and service commands

**Linux Servers:-**

1. Network Time Protocol(NTP)
2. Secure Shell(SSH)
3. Apache and NGINX servers
4. Certificate Authority(CA)
5. Domain name System(DNS)
6. Dynamic Host Configuration Protocol(DHCP)
7. Authentication Servers
8. Proxy Servers
9. Virtual Private Networks(VPN)
10. Monitoring Servers
11. Database Servers
12. Mail Servers
13. Load Balancers

**Scheduling and Automation:-**

1. cron
2. job control commands
3. kill command