# Disk and Partition Commands

## df

The df is an acronym for disk free.

The df command tells the amount of space used and available on the file system.

Use the df command for displaying available space on a file system:

1. View used space.
2. See free disk space.
3. Show what filesystems are mounted.
4. Find out if disk capacity has been reached on a partition or other device networked devices.
5. List the number of inodes still available for Linux and Unix box
6. Discover if there is sufficient space on a given partition so that developers or sysadmins can install/upgrade apps
7. Send email alerts or text messages using a shell script or Perl script when you run out of disk space.

**Syntax:**

df

df /dir

df [option] [/dev/DEVICE]

### Examples

Getting help about df command:

man df

df --help

To see information about all mounted Linux or Unix file systems, enter:

df

The df command and space can have the following info for each device name:

* **Filesystem** : The name of your file system/device.
* **1K-blocks** : The size in one-kilobyte blocks.
* **Used** : Total amount of space allocated and used by existing files in the file system (FS).
* **Avail** (**Used Available**) : Total amount of space available (free space) in file system for the creation of new files.
* **Capacity** (**Use%**) : The percentage of blocks used by the FS.
* **Mounted on** : The mount point is the directory in which it is mounted.

**Note**: Some advanced file systems and df versions can also report space reserved by the system.

**File systems info**

To display information about /boot or /rsnapshot file system, run:

df /boot

df /rsnapshot

We can display information in various block format. For instance, see 1M-byte blocks, run:

df -m

df -m /boot

For 1 Gibibyte blocks, run:

df -g

df -g /var

We can explicitly use 512 byte blocks:

df -b

df -b /usr

Pass the -k for 1024-byte blocks (default):

df -k

df -k zroot/var/log

**Human readable output**

Perhaps most important is the “Human readable” output by passing the **-h** option. It uses unit suffixes such as Byte, Kibibyte, Mebibyte, Gibibyte, Tebibyte, and Pebibyte (based on powers of 1024). You get the ease of reading as df reduces the number of digits to four or fewer. Please note the **-H** is same as **-h** but based on powers of 1000:

df -h # lowercase 'h'

df -H # UPPERCASE 'H'

df -H zroot/tmp

df -H /dev/md0

**Include File System Type**

Want to include file system type in df command output? Try passing the -T option:

df -T

df -T -H

df -T -H zroot/usr/ports

df -T -H /dev/md0

**Show statistics about the number of free and used inodes**

Pass the -i as follows:

df -i

df -i -H

df -i -H -T

## fdisk

Partition table manipulator for Linux.

**Syntax:**

fdisk [-uc] [-b sectorsize] [-C cyls] [-H heads] [-S sects] device

fdisk -l [-u] [device...]

fdisk -s partition...

fdisk -v

fdisk -h

Hard disks can be divided into one or more logical disks called partitions. This division is described in the partition table found in sector 0 of the disk.

### Options

|  |  |
| --- | --- |
| **TAG** | **DESCRIPTION** |
| **-b sectorsize** | Specify the sector size of the disk. Valid values are 512, 1024, 2048 or 4096. (Recent kernels know the sector size. Use this only on old kernels or to override the kernel's ideas.) Since util-linux-ng 2.17 fdisk differentiates between logical and physical sector size. This option changes both sector sizes to sectorsize. |
| **-h** | Print help and then exit. |
| **-c** | Switch off DOS-compatible mode. (Recommended) |
| **-C cyls** | Specify the number of cylinders of the disk. I have no idea why anybody would want to do so |
| **-H heads** | Specify the number of heads of the disk. (Not the physical number, of course, but the number used for partition tables.) Reasonable values are 255 and 16. |
| **-S sects** | Specify the number of sectors per track of the disk. (Not the physical number, of course, but the number used for partition tables.) A reasonable value is 63. |
| **-l** | List the partition tables for the specified devices and then exit. If no devices are given, those mentioned in /proc/partitions (if that exists) are used |

### Examples

**Example 1: View all Disk Partitions**

The following basic command list all existing disk partition on your system. The ‘**-l**‘ argument stand for (listing all partitions) is used with fdisk command to view all available partitions on Linux. The partitions are displayed by their device’s names. For example: **/dev/sda**, **/dev/sdb** or **/dev/sdc**.

[root@tecmint.com ~]# fdisk -l

Disk /dev/sda: 637.8 GB, 637802643456 bytes

255 heads, 63 sectors/track, 77541 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sda1 \* 1 13 104391 83 Linux

/dev/sda2 14 2624 20972857+ 83 Linux

/dev/sda3 2625 4582 15727635 83 Linux

/dev/sda4 4583 77541 586043167+ 5 Extended

/dev/sda5 4583 5887 10482381 83 Linux

/dev/sda6 5888 7192 10482381 83 Linux

/dev/sda7 7193 7845 5245191 83 Linux

/dev/sda8 7846 8367 4192933+ 82 Linux swap / Solaris

/dev/sda9 8368 77541 555640123+ 8e Linux LVM

**Example 2: View Specific Disk Partition in Linux**

To view all partitions of specific hard disk use the option ‘**-l**‘ with device name. For example, the following command will display all disk partitions of device **/dev/sda**. If you’ve different device names, simple write device name as **/dev/sdb** or **/dev/sdc**.

[root@tecmint.com ~]# fdisk -l /dev/sda

Disk /dev/sda: 637.8 GB, 637802643456 bytes

255 heads, 63 sectors/track, 77541 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sda1 \* 1 13 104391 83 Linux

/dev/sda2 14 2624 20972857+ 83 Linux

/dev/sda3 2625 4582 15727635 83 Linux

/dev/sda4 4583 77541 586043167+ 5 Extended

/dev/sda5 4583 5887 10482381 83 Linux

/dev/sda6 5888 7192 10482381 83 Linux

/dev/sda7 7193 7845 5245191 83 Linux

/dev/sda8 7846 8367 4192933+ 82 Linux swap / Solaris

/dev/sda9 8368 77541 555640123+ 8e Linux LVM

**Example 3: Check all Available fdisk Commands**

If you would like to view all commands which are available for fdisk. Simply use the following command by mentioning the hard disk name such as **/dev/sda**as shown below. The following command will give you output similar to below.

[root@tecmint ~]# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to

switch off the mode (command 'c') and change display units to

sectors (command 'u').

Command (m for help):

Type ‘**m**‘ to see the list of all available commands of fdisk which can be operated on **/dev/sda** hard disk. After, I enter ‘**m**‘ on the screen, you will see the all available options for fdisk that you can be used on the **/dev/sda** device.

[root@tecmint ~]# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to

switch off the mode (command 'c') and change display units to

sectors (command 'u').

Command (m for help): **m**

Command action

a toggle a bootable flag

b edit bsd disklabel

c toggle the dos compatibility flag

d delete a partition

l list known partition types

m print this menu

n add a new partition

o create a new empty DOS partition table

p print the partition table

q quit without saving changes

s create a new empty Sun disklabel

t change a partition's system id

u change display/entry units

v verify the partition table

w write table to disk and exit

x extra functionality (experts only)

Command (m for help):

**Example 4: Print all Partition Table**

To print all partition table of hard disk, you must be on command mode of specific hard disk say **/dev/sda**.

[root@tecmint ~]# fdisk /dev/sda

From the command mode, enter ‘**p**‘ instead of ‘**m**‘ as we did earlier. As I enter ‘**p**‘, it will print the specific **/dev/sda** partition table.

Command (m for help): p

Disk /dev/sda: 637.8 GB, 637802643456 bytes

255 heads, 63 sectors/track, 77541 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sda1 \* 1 13 104391 83 Linux

/dev/sda2 14 2624 20972857+ 83 Linux

/dev/sda3 2625 4582 15727635 83 Linux

/dev/sda4 4583 77541 586043167+ 5 Extended

/dev/sda5 4583 5887 10482381 83 Linux

/dev/sda6 5888 7192 10482381 83 Linux

/dev/sda7 7193 7845 5245191 83 Linux

/dev/sda8 7846 8367 4192933+ 82 Linux swap / Solaris

/dev/sda9 8368 77541 555640123+ 8e Linux LVM

Command (m for help):

**Example 5: How to Delete a Partition**

**Warning** : Be careful, while performing this step, because using option ‘**d**‘ will completely delete partition from system and may lost all data in partition.

If you would like to delete a specific partition (i.e **/dev/sda9**) from the specific hard disk such as **/dev/sda**. You must be in fdisk command mode to do this.

[root@tecmint ~]# fdisk /dev/sda

Next, enter ‘**d**‘ to delete any given partition name from the system. As I enter ‘**d**‘, it will prompt me to enter partition number that I want to delete from **/dev/sda** hard disk. Suppose I enter number ‘**4**‘ here, then it will delete partition number ‘**4**‘ (i.e. **/dev/sda4**) disk and shows free space in partition table. Enter ‘**w**‘ to write table to disk and exit after making new alterations to partition table. The new changes would only take place after next reboot of system. This can be easily understood from the below output.

[root@tecmint ~]# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to

switch off the mode (command 'c') and change display units to

sectors (command 'u').

Command (m for help): **d**

Partition number (1-4): **4**

Command (m for help): **w**

The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource busy.

The kernel still uses the old table. The new table will be used at

the next reboot or after you run partprobe(8) or kpartx(8)

Syncing disks.

You have new mail in /var/spool/mail/root

**Example 6: How to Create a New Partition**

If you’ve free space left on one of your device say **/dev/sda** and would like to create a new partition under it. Then you must be in fdisk command mode of **/dev/sda**. Type the following command to enter into command mode of specific hard disk.

[root@tecmint ~]# fdisk /dev/sda

After entering in command mode, now press “**n**” command to create a new partition under **/dev/sda** with specific size. This can be demonstrated with the help of following given output.

[root@tecmint ~]# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to

switch off the mode (command 'c') and change display units to

sectors (command 'u').

Command (m for help): n

Command action

e extended

p primary partition (1-4)

**e**

While creating a new partition, it will ask you two options ‘extended‘ or ‘primary‘ partition creation. Press ‘e‘ for extended partition and ‘p‘ for primary partition. Then it will ask you to enter following two inputs.

* First cylinder number of the partition to be create.
* Last cylinder number of the partition to be created (Last cylinder, +cylinders or +size).

You can enter the size of cylinder by adding “**+5000M**” in last cylinder. Here, ‘**+**‘ means addition and **5000M** means size of new partition (i.e **5000MB**). Please keep in mind that after creating a new partition, you should run ‘**w**‘ command to alter and save new changes to partition table and finally reboot your system to verify newly created partition.

Command (m for help): w

The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource busy.

The kernel still uses the old table. The new table will be used at

the next reboot or after you run partprobe(8) or kpartx(8)

Syncing disks.

**Example 7: How to Format a Partition**

After the new partition is created, don’t skip to format the newly created partition using ‘**mkfs**‘ command. Type the following command in the terminal to format a partition. Here **/dev/sda4** is my newly created partition.

[root@tecmint ~]# mkfs.ext4 /dev/sda4

**Exmaple 8: How to Check Size of a Partition**

After formatting new partition, check the size of that partition using flag ‘**s**‘ (displays size in blocks) with fdisk command. This way you can check size of any specific device.

[root@tecmint ~]# fdisk -s /dev/sda2

5194304

**Example 9: How to Fix Partition Table Order**

If you’ve deleted a logical partition and again recreated it, you might notice ‘**partition out of order**‘ problem or error message like ‘**Partition table entries are not in disk order**‘.

For example, when three logical partitions such as (**sda4**, **sda5** and **sda6**) are deleted, and new partition created, you might expect the new partition name would be **sda4**. But, the system would create it as **sda5**. This happens because of, after the partition are deleted, **sda7** partition had been moved as **sda4** and free space shift to the end.

To fix such partition order problems, and assign **sda4** to the newly created partition, issue the ‘**x**‘ to enter an extra functionality section and then enter ‘**f**‘ expert command to fix the order of partition table as shown below.

[root@tecmint ~]# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to

switch off the mode (command 'c') and change display units to

sectors (command 'u').

Command (m for help): x

Expert command (m for help): f

Done.

Expert command (m for help): w

The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource busy.

The kernel still uses the old table. The new table will be used at

the next reboot or after you run partprobe(8) or kpartx(8)

Syncing disks.

After, running ‘**f**‘ command, don’t forget to run ‘**w**‘ command to save and exit from fdisk command mode. Once it fixed partition table order, you will no longer get error messages.

**Example 10: How to Disable Boot Flag (\*) of a Partition**

By default, fdisk command shows the boot flag (i.e. ‘**\***‘) symbol on each partition. If you want to enable or disable boot flag on a specific partition, do the following steps.

[root@tecmint ~]# fdisk /dev/sda

Press ‘**p**‘ command to view the current partition table, you see there is a boot flag (asterisk (**\***) symbol in orange color) on **/dev/sda1** disk as shown below.

[root@tecmint ~]# fdisk /dev/sda

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to

switch off the mode (command 'c') and change display units to

sectors (command 'u').

Command (m for help): p

Disk /dev/sda: 637.8 GB, 637802643456 bytes

255 heads, 63 sectors/track, 77541 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sda1 **\*** 1 13 104391 83 Linux

/dev/sda2 14 2624 20972857+ 83 Linux

/dev/sda3 2625 4582 15727635 83 Linux

/dev/sda4 4583 77541 586043167+ 5 Extended

/dev/sda5 4583 5887 10482381 83 Linux

/dev/sda6 5888 7192 10482381 83 Linux

/dev/sda7 7193 7845 5245191 83 Linux

/dev/sda8 7846 8367 4192933+ 82 Linux swap / Solaris

/dev/sda9 8368 77541 555640123+ 8e Linux LVM

Next enter command ‘**a**‘ to disable boot flag, then enter partition number ‘**1**‘ as (i.e. **/dev/sda1**) in my case. This will disable boot flag on the partition **/dev/sda1**. This will remove the asterisk (**\***) flag.

Command (m for help): **a**

Partition number (1-9): **1**

Command (m for help): **p**

Disk /dev/sda: 637.8 GB, 637802643456 bytes

255 heads, 63 sectors/track, 77541 cylinders

Units = cylinders of 16065 \* 512 = 8225280 bytes

Device Boot Start End Blocks Id System

/dev/sda1 1 13 104391 83 Linux

/dev/sda2 14 2624 20972857+ 83 Linux

/dev/sda3 2625 4582 15727635 83 Linux

/dev/sda4 4583 77541 586043167+ 5 Extended

/dev/sda5 4583 5887 10482381 83 Linux

/dev/sda6 5888 7192 10482381 83 Linux

/dev/sda7 7193 7845 5245191 83 Linux

/dev/sda8 7846 8367 4192933+ 82 Linux swap / Solaris

/dev/sda9 8368 77541 555640123+ 8e Linux LVM

Command (m for help):

Example 11: Quit without saving changes

[root@tecmint ~]# fdisk /dev/sda

Enter the **q** command.

Command (m for help): q

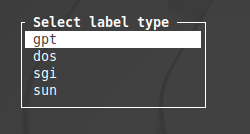
Done

## cfdisk

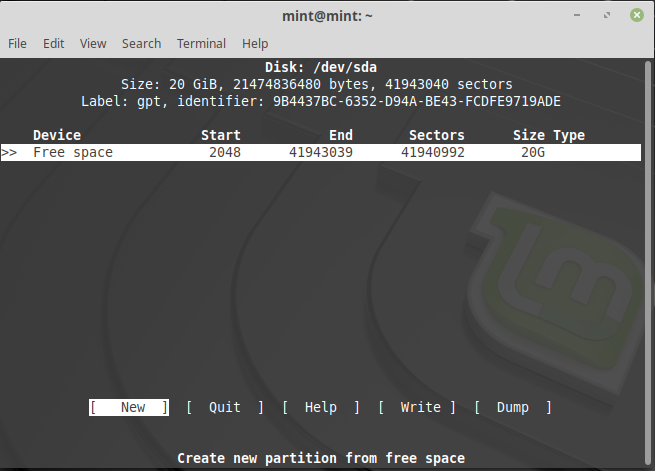
**cfdisk** command is used to create, delete, and modify partitions on a disk device. It displays or manipulates the disk partition table by providing a text-based “graphical” interface.

cfdisk /dev/sda

After running you get a prompt like this:



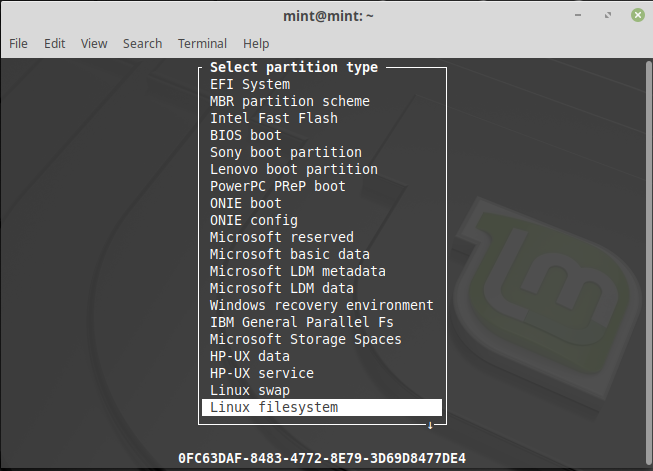
Choose *gpt*from the list. Now you will see a partition table like this:



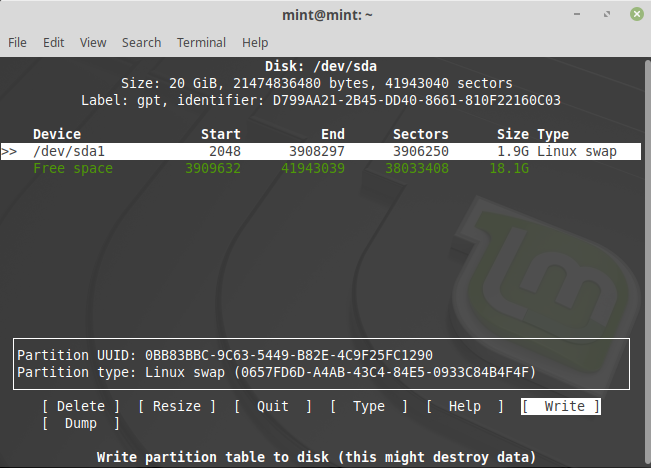
**Creating Partitions Using cfdisk:**

* See the available free space. Here we have 20 GB. Select **NEW** and create a new partition. Use up-down arrow keys to navigate and enter to select.
* You can do many things with the free space, if you are installing a new system with a command line interface, you can see an option of using the selected space as **primary partition.**

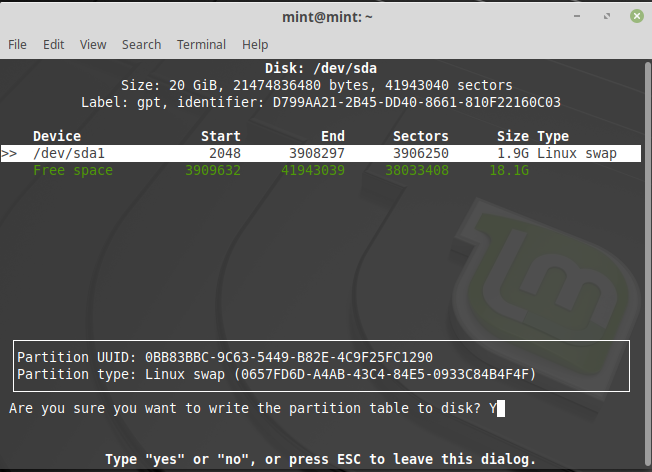
**Example:**Select the size 2GB. Enter -> and select**primary**. Similarly we can do a logical partition also.



* After sizing the partition, select what type do you want, in my case, I am choosing **Linux Swap**.
* After selecting the size and type **write**to the disk:



You will see a prompt like this:



### Options

* **-h, –help:** It displays help text and exit.
* **-L, –color[=when]:** Colorize the output. The optional argument *when*can be auto, never or always. If the *when*argument is omitted, it defaults to auto. The colors can be disabled, for the current built-in default see *–help* output. See also the COLORS section.
* **-V, –version:**Display version information and exit.
* **-z, –zero:**Start with an in-memory zeroed partition table. This option does not zero the partition table on the disk; rather, it simply starts the program without reading the existing partition table. This option allows you to create a new partition table from scratch or from a sfdisk-compatible script.

**Other command line commands:** While using **cfdisk** you can use simple commands just like we use in vi editor for saving, inserting etc. The list of commands are as follows:

* **b**: Toggle bootable flag of the current partition. It allows the user to select which partition is primary in the bootable drive. Just press **b** to see the results, no need of using **ctrl**.
* **d**: It will will delete the current marked partition, making a free space for new partition.
* **h**: Will print the help screen, showing commands used .
* **n**: Will create a new partition of the marked free space .
* **q**: Will quit the program without writing partitions to the table.
* **s**: Will fix the partitions order if they are now in proper array.
* **t**: Will allow you to change the partition type, allowing you to select from the list.
* **u**: Will dump the disk layout in a specified script file name
* **W**: Will allow the user to write the data to the disk. The user will be asked if he or she wants to write or not by simply taking input “yes” or “no”.
* **x**: Will allow the user to hide or display all extra information of the partition.
* **Up-Arrow**: Will allow the user to move the cursor to the previous partition, like moving up in the given table list.
* **Down-Arrow**: This option allows the user to move the cursor to the next partition, next partition because every new partition is placed after the previous partition.
* **Left-Arrow**: This option allows the user to enter previous menu item.
* **Right-Arrow**: This option allows the user to enter to the next menu item.

## lsblk

**Lsblk** is used to display details about block devices and these block devices(Except ram disk) are basically those files that represent devices connected to the pc. It queries **/sys** virtual file system and **udev db** to obtain information that it displays. And it basically displays output in a tree-like structure. This command comes pre-installed with the util-Linux package.

### Installing lsblk Command

Many Linux distributions do not have lsblk command pre-installed. To install it use the following commands as per your Linux distribution.

**In case of Debian/Ubuntu**

$sudo apt-get install util-linux

**In case of CentOS/RedHat**

$sudo yum install util-linux-ng

**In case of Fedora OS**

$sudo yum install util-linux-ng

### Working with lsblk command

**1.**To display block devices.

$lsblk

**2. To display empty block devices as well.**

#lsblk -a

3. To print size information in bytes.

$lsblk -b

4. To print information about device owner, group, and mode of block devices.

$lsblk -m

5. To print selected coulumns of block-devices.

$lsblk -o SIZE, NAME, MOUNTPOINT

6. To hide column headings.

$lsblk -dn

7. To display help section of the command.

$lsblk --help

## blkid

Utility to locate/print block device attributes

**Syntax:**

**blkid** [ **-hlv** ] [ [ **-c** *cachefile* ] **-w** *writecachefile* ] [ **-o** *format* ] [ **-s** *tag* ] [ **-t** *NAME*=*value* ] [ *device ...* ]

The **blkid** program can determine the type of content (e.g. filesystem, swap) a block device holds, and also attributes (tokens, NAME=value pairs) from the content metadata (e.g. LABEL or UUID fields).

**blkid** has two main forms of operation: either searching for a device with a specific NAME=value pair, or displaying NAME=value pairs for one or more devices.

### Options

|  |  |
| --- | --- |
| **Tag** | **Description** |
| -c | Read from *cachefile* instead of reading from the default cache file */etc/blkid/blkid.tab*. If you want to start with a clean cache (i.e. don’t report devices previously scanned but not necessarily available at this time), specify */dev/null.* |
| -h | Display a usage message and exit. |
| -l | Look up the device that matches the search parameter specified using the -t option, assuming that there is only one matching the search parameter. For a system with a large number of disks, this will be more efficient by avoiding the need to revalidate devices unless absolutely necessary. If this option is not specified, blkid will use a less efficient approach, which allows blkid to print all of the devices that match the search parameter. |
|  | This option is best used for tag searches such as *LABEL=data\_vol* or *UUID=e280469a-d06f-4c0b-b068-44f3b576029e*. If you want blkid to print all of the ext3 filesystems using a search parameter such as *TYPE=ext3*, then this option should *not* be used. |
| -o | Display blkid’s output using the specified format. The *format* parameter may be *full*, (the default), *value*, (only print the value of any tags printed by blkid) or *device* (only print the device name). |
| -s | Show only the tags for each (specified) device that match *tag*. It is possible to specify multiple -s options. If no tag is specified, then all tokens are shown for all (specified) devices. In order to just refresh the cache without showing any tokens use -s none with no other options. |
| -t | Search for block devices with tokens named *NAME* that have the value *value*, and display any devices which are found. Common values for *NAME* include TYPE, LABEL, and UUID. If there are no devices specified on the command line, all block devices will be searched; otherwise, only search the devices specified by the user. |
| -v | Display version number and exit. |
| -w | *<writecachefile>* Write the device cache to *writecachefile* instead of writing it to the default cache file */etc/blkid/blkid.tab*. If you don’t want to save the cache to the default file, specify */dev/null.* If not specified it will be the same file as that given by the -c option. |
| *<device>* | |
|  | Display tokens from only the specified device. It is possible to give multiple *<device>* options on the command line. If none is given, all devices which appear in */proc/partitions* are shown, if they are recognized. |

### Return Code

If the specified token was found, or if any tags were shown from (specified) devices 0 is returned. If the specified token was not found, or no (specified) devices could be identified, an exit code of 2 is returned. For usage or other errors, an exit code of 4 is returned.

## mdadm

Manage MD devices *aka* Linux Software RAID

**Syntax:**

**mdadm***[mode] <raiddevice> [options] <component-devices>*

RAID devices are virtual devices created from two or more real block devices. This allows multiple devices (typically disk drives or partitions thereof) to be combined into a single device to hold (for example) a single filesystem. Some RAID levels include redundancy and so can survive some degree of device failure.

Linux Software RAID devices are implemented through the md (Multiple Devices) device driver.

Currently, Linux supports **LINEAR** md devices, **RAID0** (striping), **RAID1** (mirroring), **RAID4**, **RAID5**, **RAID6**, **RAID10**, **MULTIPATH**, and **FAULTY**.

**MULTIPATH** is not a Software RAID mechanism, but does involve multiple devices: each device is a path to one common physical storage device.

**FAULTY** is also not true RAID, and it only involves one device. It provides a layer over a true device that can be used to inject faults.

### Modes

mdadm has several major modes of operation:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| Assemble | |
|  | Assemble the components of a previously created array into an active array. Components can be explicitly given or can be searched for. *mdadm* checks that the components do form a bona fide array, and can, on request, fiddle superblock information so as to assemble a faulty array. |
| Build | Build an array that doesn’t have per-device superblocks. For these sorts of arrays, *mdadm* cannot differentiate between initial creation and subsequent assembly of an array. It also cannot perform any checks that appropriate components have been requested. Because of this, the Build mode should only be used together with a complete understanding of what you are doing. |
| Create | Create a new array with per-device superblocks. |
| Follow or Monitor | |
|  | Monitor one or more md devices and act on any state changes. This is only meaningful for raid1, 4, 5, 6, 10 or multipath arrays, as only these have interesting state. raid0 or linear never have missing, spare, or failed drives, so there is nothing to monitor. |
| Grow | Grow (or shrink) an array, or otherwise reshape it in some way. Currently supported growth options include changing the active size of component devices and changing the number of active devices in RAID levels 1/4/5/6, as well as adding or removing a write-intent bitmap. |
| Incremental Assembly | |
|  | Add a single device to an appropriate array. If the addition of the device makes the array runnable, the array will be started. This provides a convenient interface to a *hot-plug* system. As each device is detected, *mdadm* has a chance to include it in some array as appropriate. |
| Manage | This is for doing things to specific components of an array such as adding new spares and removing faulty devices. |
| Misc | This is an ’everything else’ mode that supports operations on active arrays, operations on component devices such as erasing old superblocks, and information gathering operations. |
| Auto-detect | |
|  | This mode does not act on a specific device or array, but rather it requests the Linux Kernel to activate any auto-detected arrays. |

### Options

Options for selecting a mode are:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| -A, --assemble | Assemble a pre-existing array. |
| -B, --build | Build a legacy array without superblocks. |
| -C, --create | Create a new array. |
| -F, --follow, --monitor | Select Monitor mode. |
| -G, --grow | Change the size or shape of an active array. |
| -I, --incremental | Add a single device into an appropriate array, and possibly start the array. |
| --auto-detect | Request that the kernel starts any auto-detected arrays. This can only work if *md* is compiled into the kernel — not if it is a module. Arrays can be auto-detected by the kernel if all the components are in primary MS-DOS partitions with partition type FD. In-kernel autodetect is not recommended for new installations. Using *mdadm* to detect and assemble arrays — possibly in an *initrd* — is substantially more flexible and should be preferred. |

If a device is given before any options, or if the first option is **--add**, **--fail**, or **--remove**, then the MANAGE mode is assumed. Anything other than these will cause the **Misc** mode to be assumed.

# SSH

## Determine IP Address

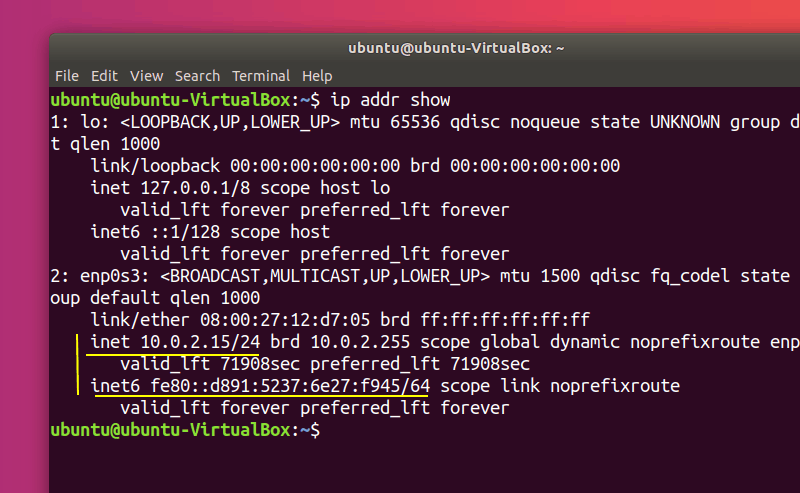
hostname

hostname -i

ip addr show

ip a

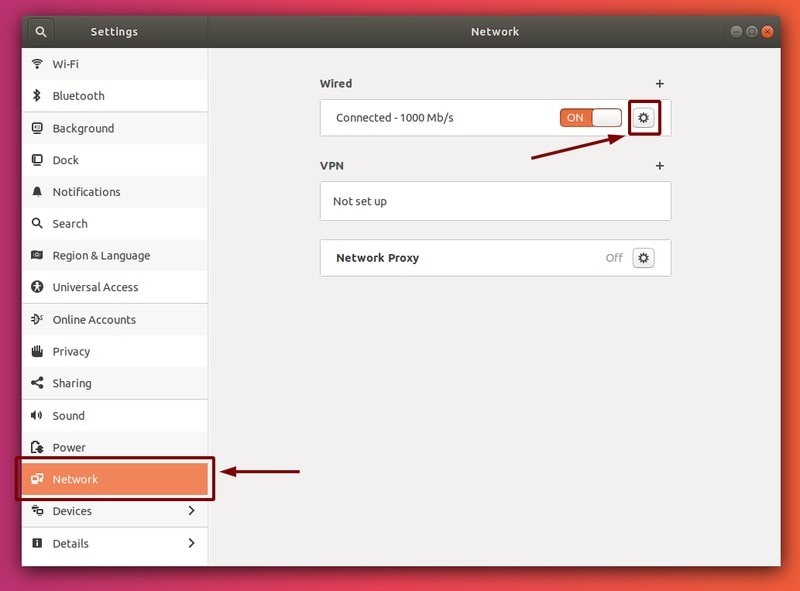
It will show you both IPv4 and IPv6 addresses:



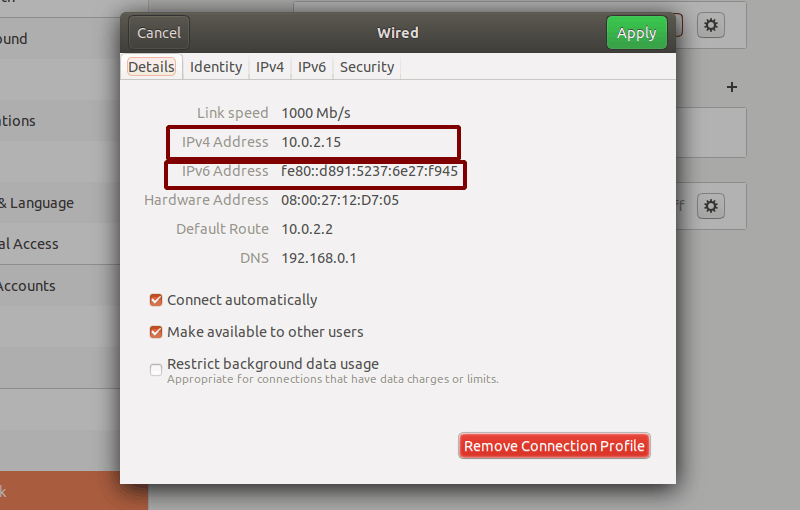
## Checking IP address in Ubuntu [GUI Method]

If you are not comfortable with the command line, you can also check IP address graphically.

Open up the Ubuntu Applications Menu (**Show Applications** in the bottom-left corner of the screen) and search for **Settings** and click on the icon, then select **Network** and then select the gear icon nest to your connection:



It should open up a window with more settings and information about your link to the network, including your IP address:



## What is SSH?

SSH, or Secure Shell, is a remote administration protocol that allows users to control and modify their remote servers over the Internet. The service was created as a secure replacement for the unencrypted Telnet and uses cryptographic techniques to ensure that all communication to and from the remote server happens in an encrypted manner. It provides a mechanism for authenticating a remote user, transferring inputs from the client to the host, and relaying the output back to the client.

The ***ssh*** or secure shell is a network protocol for operating networking services securely over a network. It uses encryption standards to securely connect and login to the remote system.

It stores a public key in the remote system and private key in the client system. These keys are produced as a pair mathematically. When both are applied to a bi-variable function, it will result in a value which will be used to check whether the pair is valid or invalid. This is the simplest explanation possible.

Any Linux or macOS user can SSH into their remote server directly from the terminal window. Windows users can take advantage ofSSH clients like **PuTTY**.  You can execute shell commands in the same manner as you would if you were physically operating the remote computer.

## How Does SSH Work

If you’re using Linux or Mac, then using SSH is very simple. If you use Windows, you will need to utilize an SSH client to open SSH connections. The most popular SSH client is PuTTY.

For Mac and Linux users, head over to your **terminal** program and then follow the procedure below:

The SSH command consists of 3 distinct parts:

ssh {user}@{host}

The SSH key command instructs your system that you want to open an encrypted Secure Shell Connection. **{user}** represents the account you want to access. For example, you may want to access the **root** user, which is basically synonymous for system administrator with complete rights to modify anything on the system. **{host}** refers to the computer you want to access. This can be an IP Address **(e.g. 244.235.23.19)** or a domain name (e.g. www.xyzdomain.com).

When you hit enter, you will be prompted to enter the password for the requested account. When you type it in, nothing will appear on the screen, but your password is, in fact being transmitted. Once you’re done typing, hit enter once again. If your password is correct, you will be greeted with a remote terminal window.

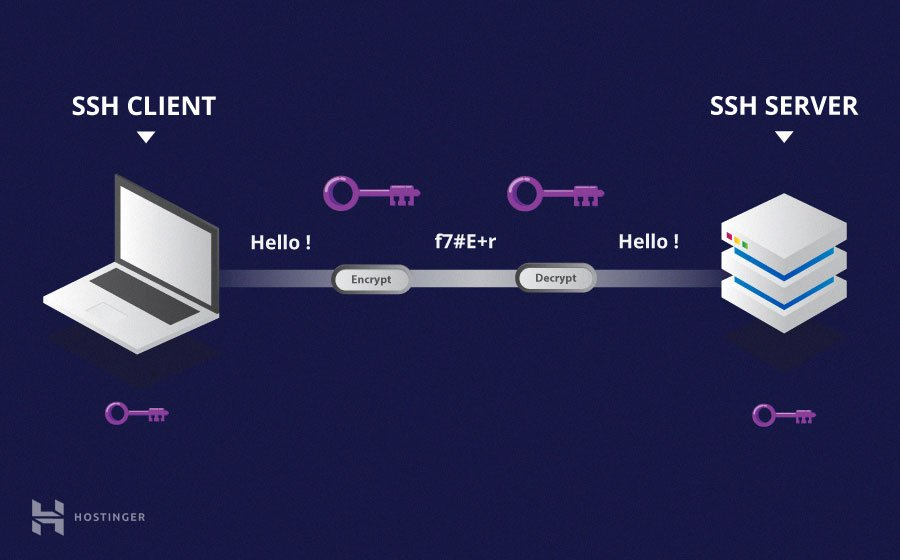
### **Understanding Different Encryption Techniques**

The significant advantage offered by SSH over its predecessors is the use of encryption to ensure secure transfer of information between the host and the client. **Host** refers to the remote server you are trying to access, while the **client** is the computer you are using to access the host. There are three different encryption technologies used by SSH:

1. Symmetrical encryption
2. Asymmetrical encryption
3. Hashing.

#### **Symmetric Encryption**

Symmetric encryption is a form of encryption where a **secret key** is used for both encryption and decryption of a message by both the client and the host. Effectively, any one possessing the key can decrypt the message being transferred.



Symmetrical encryption is often called **shared key** or **shared secret** encryption. There is usually only one key that is used, or sometimes a pair keys where one key can easily be calculated using the other key.

Symmetric keys are used to encrypt the entire communication during a SSH Session. Both the client and the server derive the secret key using an agreed method, and the resultant key is never disclosed to any third party. The process of creating a symmetric key is carried out by a **key exchange algorithm***.* What makes this algorithm particularly secure is the fact that the key is never transmitted between the client and the host. Instead, the two computers share public pieces of data and then manipulate it to independently calculate the secret key. Even if another machine captures the publically shared data, it won’t be able to calculate the key because the key exchange algorithm is not known.

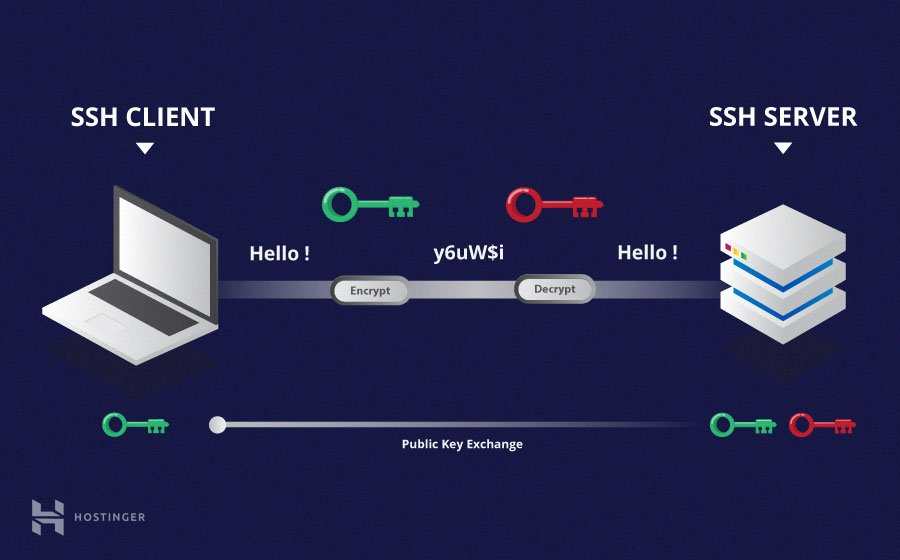
It must be noted, however, that the secret token is specific to each SSH session, and is generated prior to client authentication. Once the key has been generated, all packets moving between the two machines must be encrypted by the private key. This includes the password typed into the console by the user, so credentials are always protected from network packet sniffers.

A variety of symmetrical encryption ciphers exist, including, but not limited to, AES (Advanced Encryption Standard), CAST128, Blowfish etc. Before establishing a secured connection, the client and a host decide upon which cipher to use, by publishing a list of supported cyphers in order of preference. The most preferred cypher from the clients supported cyphers that is present on the host’s list is used as the bidirectional cypher.

For example, if two Ubuntu 14.04 LTS machines are communicating with each other over SSH, they will use **aes128-ctr** as their default cipher.

#### **Asymmetric Encryption**

Unlike symmetrical encryption, asymmetrical encryption uses two separate keys for encryption and decryption. These two keys are known as the **public key** and the **private key**. Together, both these keys form a **public-private key pair**.



The public key, as the name suggest is openly distributed and shared with all parties. While it is closely linked with the private key in terms of functionality, the private key cannot be mathematically computed from the public key. The relation between the two keys is highly complex: a message that is encrypted by a machine’s public key, can only be decrypted by the same machine’s private key. This one-way relation means that the public key cannot decrypt its own messages, nor can it decrypt anything encrypted by the private key.

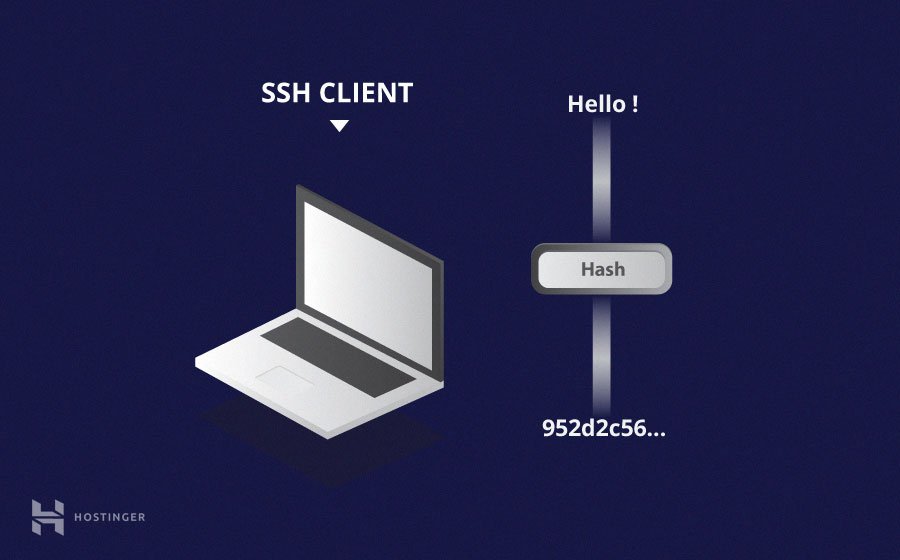
The private key must remain private i.e. for the connection to be secured, no third party must ever know it. The strength of the entire connection lies in the fact that the private key is never revealed, as it is the only component capable of decrypting messages that were encrypted using its own public key. Therefore, any party with the capability to decrypt publicly signed messages must possess the corresponding private key.

Unlike the general perception, asymmetrical encryption is not used to encrypt the entire SSH session. Instead, it is only used during the key exchange algorithm of symmetric encryption. Before initiating a secured connection, both parties generate temporary public-private key pairs, and share their respective private keys to produce the shared secret key.

Once a secured symmetric communication has been established, the server uses the clients public key to generate and challenge and transmit it to the client for authentication. If the client can successfully decrypt the message, it means that it holds the private key required for the connection. The SSH session then begins.

#### **Hashing**

One-way hashing is another form of cryptography used in Secure Shell Connections. One-way-hash functions differ from the above two forms of encryption in the sense that they are never meant to be decrypted. They generate a unique value of a fixed length for each input that shows no clear trend which can exploited. This makes them practically impossible to reverse.



It is easy to generate a cryptographic hash from a given input, but impossible to generate the input from the hash. This means that if a client holds the correct input, they can generate the crypto-graphic hash and compare its value to verify whether they possess the correct input.

SSH uses hashes to verify the authenticity of messages. This is done using HMACs, or **H**ash-based **M**essage **A**uthentication **C**odes. This ensures that the command received is not tampered with in any way.

While the symmetrical encryption algorithm is being selected, a suitable message authentication algorithm is also selected. This works in a similar way to how the cipher is selected, as explained in the symmetric encryption section.

Each message that is transmitted must contain a MAC, which is calculated using the symmetric key, packet sequence number, and the message contents. It is sent outside the symmetrically encrypted data as the concluding section of the communication packet.

### **How Does SSH Work with These Encryption Techniques**

The way SSH works is by making use of a client-server model to allow for authentication of two remote systems and encryption of the data that passes between them.

SSH operates on TCP port 22 by default (though this can be changed if needed). The host (server) listens on port 22 (or any other SSH assigned port) for incoming connections. It organizes the secure connection by authenticating the client and opening the correct shell environment if the verification is successful.



The client must begin the SSH connection by initiating the TCP handshake with the server, ensuring a secured symmetric connection, verifying whether the identity displayed by the server match previous records (typically recorded in an RSA key store file), and presenting the required user credentials to authenticate the connection.

There are two stages to establishing a connection: first both the systems must agree upon encryption standards to protect future communications, and second, the user must authenticate themselves. If the credentials match, then the user is granted access.

### **Session Encryption Negotiation**

When a client tries to connect to the server via TCP, the server presents the encryption protocols and respective versions that it supports. If the client has a similar matching pair of protocol and version, an agreement is reached and the connection is started with the accepted protocol. The server also uses an asymmetric public key which the client can use to verify the authenticity of the host.

Once this is established, the two parties use what is known as a **Diffie-Hellman Key Exchange Algorithm** to create a symmetrical key. This algorithm allows both the client and the server to arrive at a shared encryption key which will be used henceforth to encrypt the entire communication session.

Here is how the algorithm works at a very basic level:

1. Both the client and the server agree on a very large prime number, which of course does not have any factor in common. This prime number value is also known as the **seed value**.
2. Next, the two parties agree on a common encryption mechanism to generate another set of values by manipulating the seed values in a specific algorithmic manner. These mechanisms, also known as encryption generators, perform large operations on the seed. An example of such a generator is AES (Advanced Encryption Standard).
3. Both the parties independently generate another prime number. This is used as a secret private key for the interaction.
4. This newly generated private key, with the shared number and encryption algorithm (e.g. AES), is used to compute a public key which is distributed to the other computer.
5. The parties then use their personal private key, the other machine’s shared public key and the original prime number to create a final shared key. This key is independently computed by both computers but will create the same encryption key on both sides.
6. Now that both sides have a shared key, they can symmetrically encrypt the entire SSH session. The same key can be used to encrypt and decrypt messages (read: section on symmetrical encryption).

Now that the secured symmetrically encrypted session has been established, the user must be authenticated.

### **Authenticating the User**

The final stage before the user is granted access to the server is authenticating his/her credentials. For this, most SSH users use a password. The user is asked to enter the username, followed by the password. These credentials securely pass through the symmetrically encrypted tunnel, so there is no chance of them being captured by a third party.

Although passwords are encrypted, it is still not recommended to use passwords for secure connections. This is because many bots can simply brute force easy or default passwords and gain access to your account. Instead, the recommended alternative is **SSH Key Pairs**.

These are a set of asymmetric keys used to authenticate the user without the need of inputting any password.

## Generate ssh key

Websites such as GitHub and Heroku are asking for your ***ssh*** public key so that you can push/deploy code without entering a password and you don’t have such a key-pair? Don’t worry. You can generate such ssh key pair with this  command:

ssh-keygen

It will prompt for a key-location (where the key will be saved) and passphrase (i.e. password). The passphrase is optional.

By default, the ssh keys are stored in .ssh directory under your home directory.

If the key-location is ***DIR\_PATH/keypairforssh***, there will be two files

1. DIR\_PATH/keypairforssh
2. DIR\_PATH/keypairforssh.pub

***1*** is the private key file which you must not share with anyone

***2*** is the public key file which can be shared with remote systems (by means of other trusted communication such as mail, physical transfer, and other secured communication tools) and services such as Github, Heroku for the respective use cases. Be sure to check thoroughly about the service for which you are connecting.