

Top 30 Linux commands Used In DevOps



Arvind Phulare

Follow

Sep 19, 2019 · 24 min read



Linux Commands in DevOps — Edureka

Linux fundamentals and Scripting is one of the most essential skills of a DevOps Professional. Most of the companies have their environment on Linux, also many CM tools like — Puppet, Chef and Ansible have their master nodes on Linux. So in this blog, I will be covering the entire command line part which is an essential part of DevOps. The topics that we will cover here are as follows -

So let us get started,

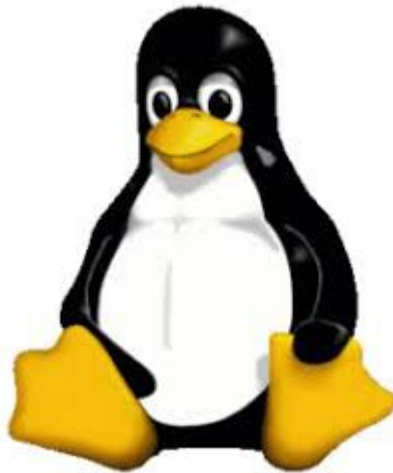
1. What is Linux?
2. Why is Linux popular?

3. Linux Commands in DevOps.

4. Shell Scripting

5. Git Commands

What is Linux?



Linux is an open-source and community-developed operating system for computers, servers, mainframes, mobile devices, and embedded devices. It has support on almost every major computer platform including x86, ARM etc, making it one of the most widely supported operating systems.

The design of Linux is similar to UNIX but it has evolved to run on a wide variety of hardware from phones to supercomputers. Every Linux-based OS contains the Linux Kernel-which manages hardware resources-and a set of software packages that make up the rest of the operating system.

Why Linux is popular?

Linux is different from the rest of the operating systems in many important aspects. Some of them are as follows

1. **Free** — First, and perhaps most importantly, Linux is free. You do not have to spend any amount to download and use it, unlike windows.

2. **Open Source** — Linux is open-source software. The code used to create Linux is free and available to the public to view, edit, and-for users with the appropriate skills-to contribute to.
3. **Secure** — Once you have Linux installed on your system, there is no need to use an antivirus! Linux is a highly secure system. Moreover, there is a global development community constantly looking at ways to enhance its security. Each upgrade makes the OS becomes more secure and robust.
4. **Stability and Performance** — Linux provides very high stability i.e. it does not need a reboot after a short period of time. Your Linux system rarely slows down or freezes.You can work without any disturbance on your Linux systems. Linux provides remarkably high performance on various networks and workstations.

Linux Commands in DevOps

In this section, we will have a look at the most frequently used Linux commands that are used while working in DevOps.

ls

This command lists all the contents in the current working directory.

syntax:

```
$ ls <flag>
```

Command	Description
<i>ls <path name></i>	By specifying the path after ls, the content in that path will be displayed
<i>ls -l</i>	Using 'l' flag, lists all the contents along with its owner settings, permissions & time stamp (long format)
<i>ls -a</i>	Using 'a' flag, lists all the hidden contents in the specified directory

```
[edureka@localhost ~]$ ls
cattest.txt  Downloads      grefile.txt  Pictures     test.txt
Desktop      eclipse        greptest.txt Public       Videos
Documents    eclipse-workspace Music         Templates
[edureka@localhost ~]$
```

sudo

This command executes only that command with root/ superuser privileges.

syntax:

```
$ sudo <command>
```

Command	Description
<i>sudo useradd <username></i>	Adding a new user
<i>sudo passwd <username></i>	Setting a password for the new user
<i>sudo userdel <username></i>	Deleting the user
<i>sudo groupadd <groupname></i>	Adding a new group
<i>sudo groupdel <groupname></i>	Deleting the group
<i>sudo usermod -g <groupname> <username></i>	Adding a user to a primary group

```
edureka@DESKTOP-IEDC4TM:~$ sudo useradd manav
[sudo] password for edureka:
edureka@DESKTOP-IEDC4TM:~$ sudo passwd manav
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
edureka@DESKTOP-IEDC4TM:~$ sudo userdel manav
edureka@DESKTOP-IEDC4TM:~$ sudo groupadd cricket
edureka@DESKTOP-IEDC4TM:~$ sudo groupdel cricket
```

cat

This command can read, modify or concatenate text files. It also displays file contents.

syntax:

```
$ cat <flag> {filename}
```

Command	Description
cat -b	This adds line numbers to non-blank lines
cat -n	This adds line numbers to all lines
cat -s	This squeezes blank lines into one line
cat -E	This shows \$ at the end of line

```
[edureka@localhost ~]$ cat cattest.txt
This file is meant to test the "concatenate" command.
[edureka@localhost ~]$ █
```

grep

This command searches for a particular string/ word in a text file. This is similar to “Ctrl+F” but executed via a CLI.

syntax:

```
$ grep <flag or element_to_search> {filename}
```

Command	Description
grep -i	Returns the results for case insensitive strings
grep -n	Returns the matching strings along with their line number
grep -v	Returns the result of lines not matching the search string
grep -c	Returns the number of lines in which the results matched the search string

```
[edureka@localhost ~]$ grep file greptest.txt
test test test file1 file2 file3 test test test
[edureka@localhost ~]$
```

sort

This command sorts the results of a search either alphabetically or numerically. It also sorts files, file contents, and directories.

syntax:

```
$ sort <flag> {filename}
```

Command	Description
<i>sort -r</i>	the flag returns the results in reverse order;
<i>sort -f</i>	the flag does case insensitive sorting
<i>sort -n</i>	the flag returns the results as per numerical order

```
[edureka@localhost Public]$ cat sorttest.txt
alias
cat
cd
rm
grep
man + help
ls
less
more
touch
mkdir
rmdir
ssh
mv
cp
[edureka@localhost Public]$ sort sorttest.txt
alias
cat
cd
cp
grep
less
ls
```

```
man + help
mkdir
more
mv
rm
rmdir
ssh
touch
[edureka@localhost Public]$
```

tail

It is complementary to head command. The tail command, as the name implies, print the last N number of data of the given input. By default, it prints the last 10 lines of the specified files. If you give more than one filename, then data from each file precedes by its file name.

syntax:

```
tail [OPTION]... [FILE]...
```

tail -n 3 state.txt or tail -3 state.txt => -n for no. of lines

tail +25 state.txt

-c num: Prints the last ‘num’ bytes from the file specified.

chown

Different users in the operating system have ownership and permission to ensure that the files are secure and put restrictions on who can modify the contents of the files. In Linux there are different users who use the system:

- Each *user* has some properties associated with them, such as a user ID and a home directory. We can add users into a group to make the process of managing users easier.
- A *group* can have zero or more users. A specified user is associated with a “default group”. It can also be a member of other groups on the system as well.

Ownership and Permissions: To protect and secure files and directory in Linux we use permissions to control what a user can do with a file or directory. Linux uses three types of permissions:

- **Read:** This permission allows the user to read files and in directories, it lets the user read directories and subdirectories stores in it.
- **Write:** This permission allows a user to modify and delete a file. Also, it allows a user to modify its contents (create, delete and rename files in it) for the directories. Unless you give the execute permission to directories, changes does not affect them.
- **Execute:** The write permission on a file executes the file. For example, if we have a file named *sh* so unless we don't give it execute permission it won't run.

Types of file Permissions:

- **User:** This type of file permission affects the owner of the file.
- **Group:** This type of file permission affects the group which owns the file. Instead of the group permissions, the user permissions will apply if the owner user is in this group.
- **Other:** This type of file permission affects all other users on the system.

Note: To view the permissions we use:

```
ls -l
```

chown command is used to change the file Owner or group. Whenever you want to change ownership you can use chown command.

Syntax:

```
chown [OPTION]... [OWNER] [:[GROUP]] FILE...
```

```
chown [OPTION]... -reference=RFILE FILE...
```


Example: To change owner of the file:

```
chown owner_name file_name
```

```
chown master file1.txt
```

where the *master* is another user in the system. Assume that if you are user named user1 and you want to change ownership to root (where your current directory is user1). use “sudo” before syntax.

```
sudo chown root file1.txt
```

chmod

This command is used to change the access permissions of files and directories.

Syntax:

```
chmod <permissions of user,group,others> {filename}
```

4 — read permission

2 — write permission

1 — execute permission

0 — no permission

```
[edureka@localhost ~]$ cat chmodtest.sh
#!/bin/sh

echo "This file is to test file permissions."
[edureka@localhost ~]$
```

```
[edureka@localhost ~]$ nano chmodtest.sh
[edureka@localhost ~]$ ./chmodtest.sh
bash: ./chmodtest.sh: Permission denied
```

```
[edureka@localhost ~]$ chmod 777 chmodtest.sh
[edureka@localhost ~]$ ./chmodtest.sh
This file is to test file permissions.
[edureka@localhost ~]$
```

lsof

While working in Linux/Unix system there might be several file and folder which are being used, some of them would be visible and some not. **lsof** command stands for **List Of Open File**. This command provides a list of files that are opened. Basically, it gives the information to find out the files which are opened by which process. With one go it lists out all open files in the output console.

Syntax:

```
$lsof [option][user name]
```

Options with Examples:

- **List all open files:** This command lists out all the files that are opened by any process in the system.

```
~$ lsof
```

- Here, you observe there are details of the opened files. ProcessId, the user associated with the process, FD(file descriptor), size of the file all together gives detailed information about the file opened by the command, process ID, user, its size, etc.
- **FD:** represents as File descriptor.
- **cwd:** Current working directory.
- **txt:** Text file.
- **mem:** Memory file.
- **mmap:** Memory-mapped device.

List all files opened by a user: There are several users of a system and each user has different requirements and accordingly they use files and devices. To find a list of files that are opened by a specific user this command is useful.

Syntax:

```
lsof -u username
```

Along with that we can see the type of file here and they are:

- **DIR:** Directory
- **REG:** Regular file
- **CHR:** Character special file

```
edureka@DESKTOP-IEDC4TM:~$ lsof
COMMAND PID TID   USER  FD   TYPE DEVICE        SIZE      NODE NAME
init      1          root   cwd   DIR    0,2    4096 2251799814207480 /
init      1          root   rtd   DIR    0,2    4096 2251799814207480 /
init      1          root   txt   REG    0,2  591344 5910974511204684 /init
init      1          root   mem   REG    0,0                280908 /init (path dev=0,2, inode=5910974511204684)
init      1          root   NOFD                /proc/1/fd (opendir: Permission denied)
init      1    5      root   cwd   DIR    0,2    4096 2251799814207480 /
init      1    5      root   rtd   DIR    0,2    4096 2251799814207480 /
init      1    5      root   txt   REG    0,2  591344 5910974511204684 /init
init      1    5      root   mem   REG    0,0                280908 /init (path dev=0,2, inode=5910974511204684)
init      1    5      root   NOFD                /proc/1/task/5/fd (opendir: Permission denied)
init      6          root   cwd   DIR    0,2    4096 2251799814207480 /
init      6          root   rtd   DIR    0,2    4096 2251799814207480 /
init      6          root   txt   REG    0,2  591344 5910974511204684 /init
init      6          root   mem   REG    0,0                280908 /init (path dev=0,2, inode=5910974511204684)
init      6          root   NOFD                /proc/6/fd (opendir: Permission denied)
bash      7      edureka   cwd   DIR    0,2    4096  844424930656251 /home/edureka
bash      7      edureka   rtd   DIR    0,2    4096 2251799814207480 /
```

ifconfig

ifconfig(interface configuration) command is used to configure the kernel-resident network interfaces. It is used at the boot time to set up the interfaces as necessary. After that, it is usually used when needed during debugging or when you need system tuning. Also, this command is used to assign the IP address and netmask to an interface or to enable or disable a given interface.

Syntax:

```
ifconfig [...OPTIONS] [INTERFACE]
```

Options:

- **-a** : This option is used to display all the interfaces available, even if they are down.

Syntax:

```
ifconfig -a
```

-s : Display a short list, instead of details.

Syntax:

```
ifconfig -s
```

id

id command in Linux is used to find out user and group names and numeric ID's (UID or group ID) of the current user or any other user in the server. This command is useful to find out the following information as listed below:

- User name and real user id.
- Find out the specific Users UID.
- Show the UID and all groups associated with a user.
- List out all the groups a user belongs to.
- Display security context of the current user.

Syntax:

```
id [OPTION]... [USER]
```

Options:

- **-g**: Print only the effective group id.
- **-G**: Print all Group ID's.
- **-n**: Prints name instead of a number.
- **-r**: Prints real ID instead of numbers.
- **-u**: Prints only the effective user ID.
- **-help**: Display help messages and exit.
- **-version**: Display the version information and exit.

Note: Without any OPTION it prints every set of identified information i.e. numeric IDs.

Examples:

- **To print your own id without any Options:**

```
id
```

The output shows the ID of current user UID and GID.

```
root@master-VirtualBox:~# id
uid=0(root) gid=0(root) groups=0(root)
root@master-VirtualBox:~#
```

- **Find a specific users id:** Now assume that we have a user named master, to find his UID we will use the command:

```
id -u master
```

```
root@master-VirtualBox:~# id -u master
1000
root@master-VirtualBox:~#
```

- **Get a specific users GID:** Again assuming to find GID of master, we will use the command:

```
id -g master
```

```
root@master-VirtualBox:~# id -g master
1000
root@master-VirtualBox:~#
```

- **Know the UID and all groups associated with a username:** In this case, we will use the user “master” to find UID and all groups associated with it, use the command:

```
id master
```

```
root@master-VirtualBox:~# id master
uid=1000(master) gid=1000(master) groups=1000(master),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),118(lpadmin),128(sambashare)
root@master-VirtualBox:~#
```

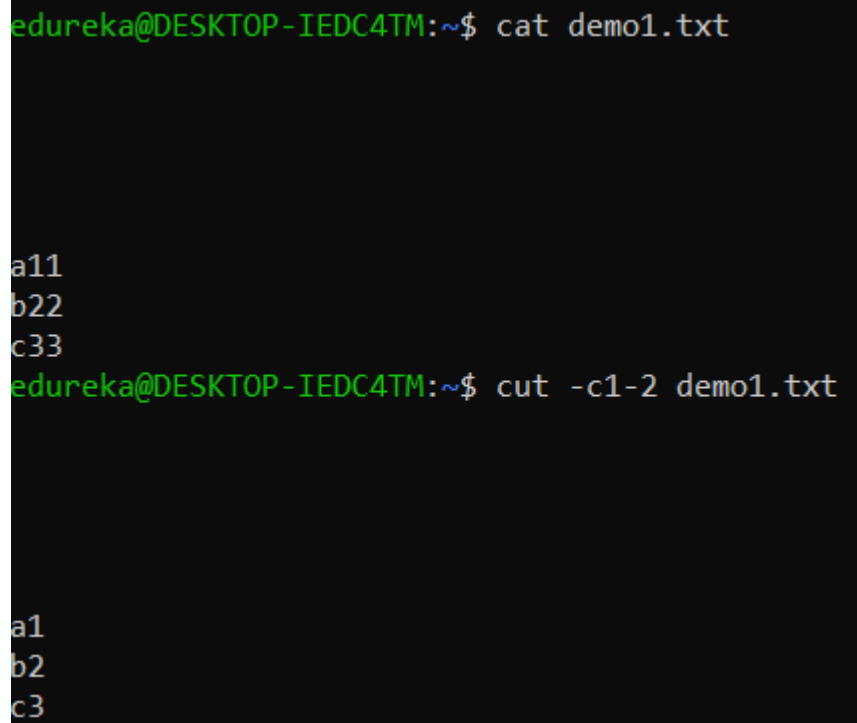
- **To find out all the groups a user belongs to:** Displaying the UID and all groups a user “master” belongs to:

```
id -G master
```

cut

Cut command is used for extracting a portion of a file using columns and delimiters. If you want to list everything in a selected column, use the “-c” flag with cut command. For example, lets select the first two columns from our demo1.txt file.

```
cut -c1-2 demo1.txt
```

A terminal window with a black background and green text. The prompt is 'edureka@DESKTOP-IEDC4TM:~\$'. The first command is 'cat demo1.txt', which outputs 'a11', 'b22', and 'c33' on three separate lines. The second command is 'cut -c1-2 demo1.txt', which outputs 'a1', 'b2', and 'c3' on three separate lines.

```
edureka@DESKTOP-IEDC4TM:~$ cat demo1.txt
a11
b22
c33
edureka@DESKTOP-IEDC4TM:~$ cut -c1-2 demo1.txt
a1
b2
c3
```

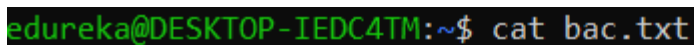
sed

Sed is a text-editor that can perform editing operations in a non-interactive way. The sed command gets its input from standard input or a file to perform the editing operation on a file. Sed is a very powerful utility and you can do a lot of file manipulations using sed. I will explain the important operation you might want to do with a text file.

If you want to replace a text in a file by searching it in a file, you can use the sed command with a substitute “s” flag to search for the specific pattern and change it.

For example, lets replace “mikesh” in test.txt file to “Mukesh”

```
sed 's/mikesh/mukesh/' test.txt
```

A terminal window with a black background and green text. The prompt is 'edureka@DESKTOP-IEDC4TM:~\$'. The command is 'cat bac.txt'.

```
edureka@DESKTOP-IEDC4TM:~$ cat bac.txt
```

```
How are u Gabbar?  
edureka@DESKTOP-IEDC4TM:~$ sed 's/Gabbar/Gabru/' bac.txt  
  
How are u Gabru?  
edureka@DESKTOP-IEDC4TM:~$
```

diff

diff command is used to find the difference between two files. This command analyses the files and prints the lines which are not similar. Lets say we have two files test and test1. you can find the difference between the two files using the following command.

Syntax –

```
diff test.txt test1.txt
```

```
edureka@DESKTOP-IEDC4TM:~$ cat test.txt  
hi, how are u?  
edureka@DESKTOP-IEDC4TM:~$ cat test1.txt  
I am fine  
  
edureka@DESKTOP-IEDC4TM:~$ diff test.txt test1.txt  
2c2,3  
< hi, how are u?  
---  
> I am fine  
>
```

history

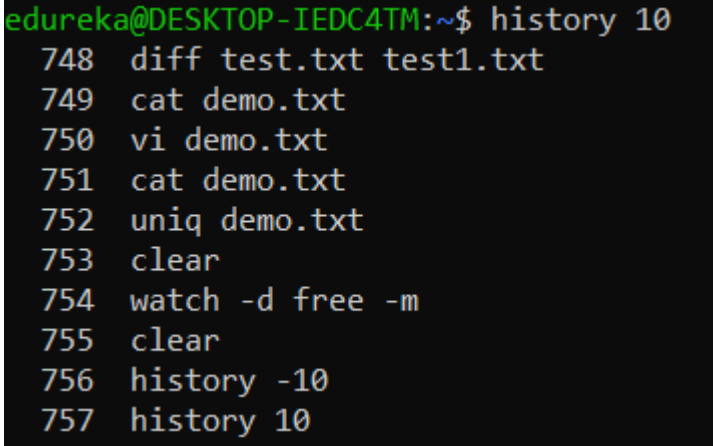
history command is used to view the previously executed command. This feature was not available in the Bourne shell. Bash and Korn support this feature in which every command executed is treated as the event and is associated with an event number using which they can be recalled and changed if required. These commands are saved in a history file. In Bash shell **history** command shows the whole list of the command.

Syntax:


```
$ history
```

To show the limited number of commands that executed previously as follows:

```
$ history 10
```

A terminal window with a black background and green text. The prompt is 'edureka@DESKTOP-IEDC4TM:~\$'. The command 'history 10' has been entered, and the output shows a list of 10 commands with their corresponding line numbers from the history. The commands are: 748 diff test.txt test1.txt, 749 cat demo.txt, 750 vi demo.txt, 751 cat demo.txt, 752 uniq demo.txt, 753 clear, 754 watch -d free -m, 755 clear, 756 history -10, and 757 history 10.

```
edureka@DESKTOP-IEDC4TM:~$ history 10
748  diff test.txt test1.txt
749  cat demo.txt
750  vi demo.txt
751  cat demo.txt
752  uniq demo.txt
753  clear
754  watch -d free -m
755  clear
756  history -10
757  history 10
```

dd

dd is a command-line utility for Unix and Unix-like operating systems whose primary purpose is to convert and copy files.

- On Unix, device drivers for hardware (such as hard disk drives) and special device files (such as `/dev/zero` and `/dev/random`) appear in the file system just like normal files.
- `dd` can also read and/or write from/to these files, provided that function is implemented in their respective drivers
- As a result, `dd` can be used for tasks such as backing up the boot sector of a hard drive, and obtaining a fixed amount of random data.
- The `dd` program can also perform conversions on the data as it is copied, including byte order swapping and conversion to and from the ASCII and EBCDIC text encodings.

Usage : The command line syntax of dd differs from many other Unix programs, in that it uses the syntax *option=value* for its command line options, rather than the more-standard *-option value* or *-option=value* formats. By default, dd reads from stdin and writes to stdout, but these can be changed by using the if (input file) and of (output file) options.

Some practical examples on dd command :

1. **To back up the entire hard disk :** To backup an entire copy of a hard disk to another hard disk connected to the same system, execute the dd command as shown. In this dd command example, the UNIX device name of the source hard disk is /dev/hda, and device name of the target hard disk is /dev/hdb.

```
# dd if = /dev/sda of = /dev/sdb
```

- “if” represents input file, and “of” represents output file. So the exact copy of /dev/sda will be available in /dev/sdb.
- If there are any errors, the above command will fail. If you give the parameter “conv=noerror” then it will continue to copy if there are read errors.
- Input file and output file should be mentioned very carefully. Just in case, you mention source device in the target and vice versa, you might lose all your data.

find

The **find** command in UNIX is a command-line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the ‘-exec’ other UNIX commands can be executed on files or folders found.

Syntax :

```
$ find [where to start searching from]
```

```
[expression determines what to find] [-options] [what to find]
```

Options :

- **-exec CMD**:The file being searched which meets the above criteria and returns 0 for as its exit status for successful command execution.
- **-ok CMD** :It works same as -exec except the user is prompted first.
- **-inum N** :Search for files with inode number 'N'.
- **-links N** :Search for files with 'N' links.

free

In LINUX, there exists a command-line utility for this and that is **free** command which displays the total amount of free space available along with the amount of memory used and swap memory in the system, and also the buffers used by the kernel.

This is pretty much what free command does for you.

Syntax:

```
$free [OPTION]
```

OPTION: refers to the options compatible with free command.

As free displays the details of the memory-related to your system, its syntax doesn't need any arguments to be passed but only options which you can use according to your wish.

Using free Command

You can use the free command as:

```
$free
```

```
edureka@DESKTOP-IEDC4TM:~$ free
              total        used        free      shared  buff/cache   available
Mem:          8303844       6516264       1558228        17720       229352     1653848
Swap:        25165824       1609448       23556376
```

/*free command without any

option shows the used

and free space of swap

and physical memory in **KB** */

When no option is used then free command produces the columnar output as shown above where column:

1. **total displays** the total installed memory (MemTotal and SwapTotal e present in /proc/meminfo).
2. **used displays** the used memory.
3. **free displays** the unused memory.
4. **shared displays** the memory used by tmpfs(Shmen e present in /proc/meminfo and displays zero in case not available).
5. **buffers displays** the memory used by kernel buffers.
6. **cache displays** the memory used by the page cache and slabs(Cached and Slab available in /proc/meminfo).
7. **buffers/cache displays** the sum of buffers and cache.

Options for free command

- **-b, — -bytes** :It displays the memory in bytes.
- **-k, — -kilo** :It displays the amount of memory in kilobytes(default).
- **-m, — -mega** :It displays the amount of memory in megabytes.
- **-g, — -giga** :It displays the amount of memory in gigabytes

ssh-keygen

Use the ssh-keygen command to generate a public/private authentication key pair. Authentication keys allow a user to connect to a remote system without supplying a

password. Keys must be generated for each user separately. If you generate key pairs as the root user, only the root can use the keys.

The following example creates the public and private parts of an RSA key:

```
ssh-keygen -t rsa
```

```
edureka@DESKTOP-IEDC4TM:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/edureka/.ssh/id_rsa):
/home/edureka/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/edureka/.ssh/id_rsa.
Your public key has been saved in /home/edureka/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:M5M+YY1m4J/XLI5KlbVj8tiZqdyDuNcIvTSdpJgLYzA edureka@DESKTOP-IEDC4TM
The key's randomart image is:
+---[RSA 2048]-----+
|
|  .  .
| E . . *..
| o . +S+=.
| + +B=%oB
| . o.=**O o
| .o.=Bo.
| o++ o.
|
+---[SHA256]-----+
```

Use the `-t` option to specify the type of key to create. Possible values are “**rsa1**” for protocol version 1, and “**dsa**”, “**ecdsa**”, or “**rsa**” for protocol version 2.

You have the option of specifying a passphrase to encrypt the private part of the key. If you encrypt your personal key, you must supply the passphrase each time you use the key. This prevents an attacker, who has access to your private key and can impersonate you and access all the computers you have access to, from being able to do so. The attacker still needs to supply the passphrase.

ip

ip command in Linux is present in the net-tools which is used for performing several network administration tasks. This command is used to show or manipulate routing, devices, and tunnels. This command is used to perform several tasks like assigning an address to a network interface or configuring network interface parameters. It can perform several other tasks like configuring and modifying the default and static routing, setting up a tunnel over IP, listing IP addresses and property information, modifying the status of the interface, assigning, deleting and setting up IP addresses and routes.

Syntax:

```
ip [ OPTIONS ] OBJECT { COMMAND | help }
```

Options:

- **address:** This option is used to show all IP addresses associated with all network devices.

```
ip address
```

```
edureka@DESKTOP-IEDC4TM:~$ ip address
20: eth0: <> mtu 1500 group default qlen 1
    link/ether b4:b6:86:26:e5:a6
    inet 169.254.59.188/16 brd 169.254.255.255 scope global dynamic
        valid_lft forever preferred_lft forever
    inet6 fe80::c48a:1a91:db03:3bbc/64 scope link dynamic
        valid_lft forever preferred_lft forever
1: lo: <LOOPBACK,UP> mtu 1500 group default qlen 1
    link/loopback 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope global dynamic
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host dynamic
        valid_lft forever preferred_lft forever
17: wifi0: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ieee802.11 34:41:5d:15:41:6d
    inet 192.168.3.245/22 brd 192.168.3.255 scope global dynamic
        valid_lft 79324sec preferred_lft 79324sec
    inet6 fe80::c20:c751:e24f:ffe5/64 scope link dynamic
        valid_lft forever preferred_lft forever
```

```

    valid_lft forever preferred_lft forever
19: wifi1: <> mtu 1500 group default qlen 1
    link/ieee802.11 34:41:5d:15:41:6e
    inet 169.254.186.128/16 brd 169.254.255.255 scope global dynamic
        valid_lft forever preferred_lft forever
    inet6 fe80::40cb:5b50:9459:ba80/64 scope link dynamic
        valid_lft forever preferred_lft forever
3: wifi2: <> mtu 1500 group default qlen 1
    link/ieee802.11 36:41:5d:15:41:6d
    inet 169.254.81.159/16 brd 169.254.255.255 scope global dynamic
        valid_lft forever preferred_lft forever
    inet6 fe80::f08c:c6a6:dd0a:519f/64 scope link dynamic
        valid_lft forever preferred_lft forever

```

- **link:** It is used to display link-layer information, it will fetch characteristics of the link-layer devices currently available. Any networking device which has a driver loaded can be classified as an available device.

ip link

```

edureka@DESKTOP-IEDC4TM:~$ ip link
20: eth0: <> mtu 1500 group default qlen 1
    link/ether b4:b6:86:26:e5:a6
1: lo: <LOOPBACK,UP> mtu 1500 group default qlen 1
    link/loopback 00:00:00:00:00:00
17: wifi0: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ieee802.11 34:41:5d:15:41:6d
19: wifi1: <> mtu 1500 group default qlen 1
    link/ieee802.11 34:41:5d:15:41:6e
3: wifi2: <> mtu 1500 group default qlen 1
    link/ieee802.11 36:41:5d:15:41:6d

```

nslookup

Nslookup (stands for “Name Server Lookup”) is a useful command for getting information from DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS related problems.

Syntax:

```
nslookup [option]
```

Options of nslookup command:

```
nslookup google.com:
```

nslookup followed by the domain name will display the “A Record” (IP Address) of the domain. Use this command to find the address record for a domain. It queries to domain name servers and get the details.

```
edureka@DESKTOP-IEDC4TM:~$ nslookup google.com
Server:      192.168.1.1
Address:     192.168.1.1#53

Non-authoritative answer:
Name:   google.com
Address: 216.58.199.174
Name:   google.com
Address: 2404:6800:4007:80c::200e
```

curl

curl is a command-line tool to transfer data to or from a server, using any of the supported protocols (HTTP, FTP, IMAP, POP3, SCP, SFTP, SMTP, TFTP, TELNET, LDAP or FILE). This command is powered by Libcurl. This tool is preferred for automation since it is designed to work without user interaction. It can transfer multiple file at once.

Syntax:

```
curl [options] [URL...]
```

The most basic uses of curl is typing the command followed by the URL.

```
curl https://www.python.org
```


-o : saves the downloaded file on the local machine with the name provided in the parameters.

Syntax:

```
curl -o [file_name] [URL...]
```

Example:

```
curl -o hello.zip ftp://speedtest.tele2.net/1MB.zip
```

tr

The **tr** command in UNIX is a command-line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace. It can be used with UNIX pipes to support more complex translation. **tr stands for translate.**

Syntax:

```
$ tr [flag] SET1 [SET2]
```

Options

- c : complements the set of characters in string.i.e., operations apply to characters not in the given set
- d : delete characters in the first set from the output.
- s : replaces repeated characters listed in the set1 with single occurrence
- t : truncates set1

Sample Commands

1. How to convert lower case to upper case

To convert from lower case to upper case the predefined sets in **tr** can be used.

```
edureka@DESKTOP-IEDC4TM:~$ cat bac.txt

How are u Gabbar?
edureka@DESKTOP-IEDC4TM:~$ cat bac.txt | tr "[a-z]" "[A-Z]"

HOW ARE U GABBAR?
```

iptables

The **iptables** is a command-line interface used to set up and maintain tables for the Netfilter firewall for IPv4, included in the Linux kernel. The firewall matches packets with rules defined in these tables and then takes the specified action on a possible match.

- *Tables* is the name for a set of chains.
- *Chain* is a collection of rules.
- *Rule* is a condition used to match packet.
- *Target* is action taken when a possible rule matches. Examples of the target are ACCEPT, DROP, QUEUE.
- *Policy* is the default action taken in case of no match with the inbuilt chains and can be ACCEPT or DROP.

Syntax:

```
iptables --table TABLE -A/-C/-D... CHAIN rule --jump Target
```

apt-get

apt-get is a command-line tool that helps in handling packages in Linux. Its main task is to retrieve the information and packages from the authenticated sources for installation,

upgrade and removal of packages along with their dependencies. Here APT stands for the *Advanced Packaging Tool*.

Syntax:

```
apt-get [options] command
```

update: This command is used to synchronize the package index files from their sources again. You need to perform an update before you upgrade.

Syntax:

```
apt-get update
```

df,du

The `df` (*disk free*) command reports the amount of available disk space being used by file systems. The `du` (*disk usage*) command reports the sizes of directory trees inclusive of all of their contents and the sizes of individual files.

The aim is to make sure you are not overshooting the 80% threshold. If you exceed the threshold it's time to scale or clean-up the mess, because running out of resources you have the change your application show some fickle behavior.

To check in a human-readable format:

```
$ sudo df -h
```

```
edureka@DESKTOP-IEDC4TM:~$ sudo df -h
[sudo] password for edureka:
Filesystem      Size  Used Avail Use% Mounted on
rootfs          443G  159G  284G  36% /
none            443G  159G  284G  36% /dev
none            443G  159G  284G  36% /run
none            443G  159G  284G  36% /run/lock
none            443G  159G  284G  36% /run/shm
```

```
none          443G  159G  284G  36% /run/user
cgroup        443G  159G  284G  36% /sys/fs/cgroup
C:\           443G  159G  284G  36% /mnt/c
E:\           489G  4.8G  484G   1% /mnt/e
```

But in most cases, you want to check which part of your system is consuming lots of disk space. Use the following command:

```
$ sudo du -h -d 1 /var/
```

```
edureka@DESKTOP-IEDC4TM:~$ sudo du -h -d 1 /var/
0          /var/backups
129M       /var/cache
0          /var/crash
132M       /var/lib
0          /var/local
676K       /var/log
0          /var/mail
0          /var/opt
0          /var/snap
0          /var/spool
0          /var/tmp
261M       /var/
```

htop

htop command in the Linux system is a command-line utility that allows the user to interactively monitor the system's vital resources or server's processes in real-time. This is a newer program compared to top command, and it offers many improvements over top command. It supports mouse operation, uses color in its output and gives visual indications about processor, memory and swap usage. htop also prints full command lines for processes and allows one to scroll both vertically and horizontally for processes and command lines respectively.

Syntax

```
htop <flag>
```

- **-d –delay** :Used to show the delay between updates, in tenths of seconds.
- **-C –no-color –no-colour** : Start htop in monochrome mode.
- **-h –help** :Used to display the help message and exit.
- **-u –user=USERNAME** :Used to show only the processes of a given user.

ps

Every process in Linux has a unique ID and can be seen using the command `ps`.

- `$ sudo ps aux`
- **a** = show processes for all users
- **u** = display the process's user/owner
- **x** = also show processes not attached to a terminal

```
edureka@DESKTOP-IEDC4TM:~$ sudo ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	8892	116	?	Ssl	Sep16	0:00	/init ro
root	6	0.0	0.0	8900	68	tty1	Ss	Sep16	0:00	/init ro
edureka	7	0.0	0.0	17040	880	tty1	S	Sep16	0:00	-bash
root	487	0.0	0.0	17272	584	tty1	S	Sep16	0:00	sudo bash
root	488	0.0	0.0	16940	1008	tty1	S	Sep16	0:00	bash
root	497	0.0	0.0	16520	484	tty1	S	Sep16	0:00	su edureka
edureka	498	0.0	0.0	16944	1752	tty1	S	Sep16	0:00	bash
root	650	0.0	0.0	17024	2368	tty1	S	12:25	0:00	sudo ps aux
root	651	0.0	0.0	17380	1916	tty1	R	12:25	0:00	ps aux

kill

`kill` command in Linux (located in `/bin/kill`), is a built-in command which is used to terminate processes manually. This command sends a signal to a process that terminates the process. If the user doesn't specify any signal which is to be sent along with `kill` command then default `TERM` signal is sent that terminates the process.

`kill -l` :To display all the available signals you can use below command option:

Syntax: `$kill -l`

```
edureka@DESKTOP-IEDC4TM:~$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT    4) SIGILL
 5) SIGTRAP     6) SIGABRT     7) SIGBUS     8) SIGFPE     9) SIGKILL
10) SIGUSR1     11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM
15) SIGTERM     16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP
20) SIGTSTP     21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM  27) SIGPROF    28) SIGWINCH
29) SIGIO       30) SIGPWR     31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1
36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6
41) SIGRTMIN+7  42) SIGRTMIN+8  43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11
46) SIGRTMIN+12 47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9
56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4
61) SIGRTMAX-3  62) SIGRTMAX-2  63) SIGRTMAX-1  64) SIGRTMAX
```

- Negative PID values are used to indicate the process group ID. If you pass a process group ID then all the processes within that group will receive the signal.
- A PID of -1 is very special as it indicates all the processes except kill and init, which is the parent process of all processes on the system.
- To display a list of running processes use the command *ps* and this will show you running processes with their PID number. To specify which process should receive the kill signal we need to provide the PID.

Syntax:

\$ps

kill pid: To show how to use a *PID* with the *kill* command.

Syntax:

```
$kill pid
```

telnet

Telnet helps to –

- connect to a remote Linux computer
- run programs remotely and conduct administration

Syntax

- telnet hostname="" or=""
- Example:
- *telnet localhost*

Shell Scripting

What is Shell?

An Operating system contains many components, but its two prime components are the Kernel and the Shell.

You can consider a Kernel as a nucleus of a computer. It makes communication between the hardware and software possible. The Kernel is the innermost part of an operating system whereas a shell is the outermost one.

A shell in a Linux operating system takes input from the user in the form of commands, processes it, and then gives an output. It acts as an interface through which a user works on the programs, commands, and scripts. A terminal accesses the shell and also runs the commands.

When the terminal is run, the Shell issues a command prompt(usually \$)where it is possible to type your input, after which the terminal executes it when you hit the Enter key. The terminal then displays the output of your commands.

The Shell wraps as a covering around the delicate interior of an Operating system protecting it from accidental damage. Hence the name is Shell.

There are two main shells in Linux:

1. **The Bourne Shell:** The prompt for this shell is \$ and its derivatives are as follows:
 - POSIX shell also is known as sh
 - Korn Shell also knew as sh
 - Bourne Again SHell is also known as bash (most popular)
2. **The C shell:** % denotes the prompt for this shell and its subcategories are as follows:
 - C shell also is known as csh
 - Tops C shell is also known as tcsh

What is Shell Scripting?

Shell scripting is writing a series of commands for the shell that can be executed. It can combine both lengthy and repetitive sequences of commands into a single and simple script. You can store this script and execute it whenever you want. This significantly reduces the effort required by the end-user.

Following are the steps to create a Shell Script –

- Create a file using a text editor such as the vi or any other editor. Name script file with extension .sh
- Start the script with `#!/bin/sh`
- Write some code.
- Save the script file as filename.sh
- For executing the script type `bash filename.sh`

“#!” is an operator called shebang that points the script to the interpreter location. So, if we use “#!/bin/sh” the script points to the bourne-shell.

We will now create a file using an editor like vi and save it with .sh extension. Copy the following program that adds and prints the sum of digits of a number entered by the user. Then run this program using the command `bash filename.sh`.

```
#!/bin/sh

echo "Enter a number"
read Num
g=$Num

# store the sum of
# digits
s=0

# use while loop to
# calculate the sum
# of all digits
while [ $Num -gt 0 ]
do
# get Remainder
k=$(( $Num % 10 ))

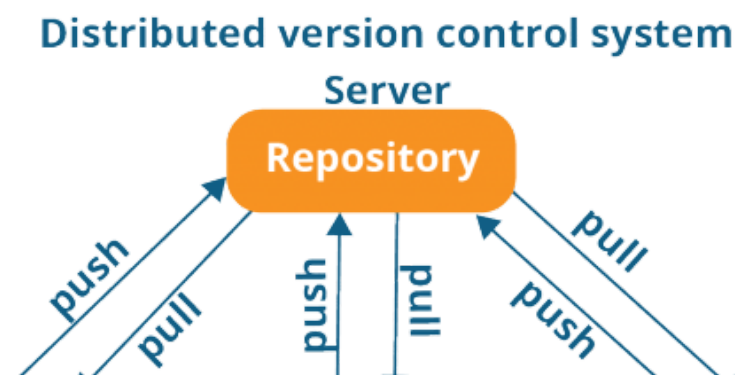
# get next digit
Num=$(( $Num / 10 ))

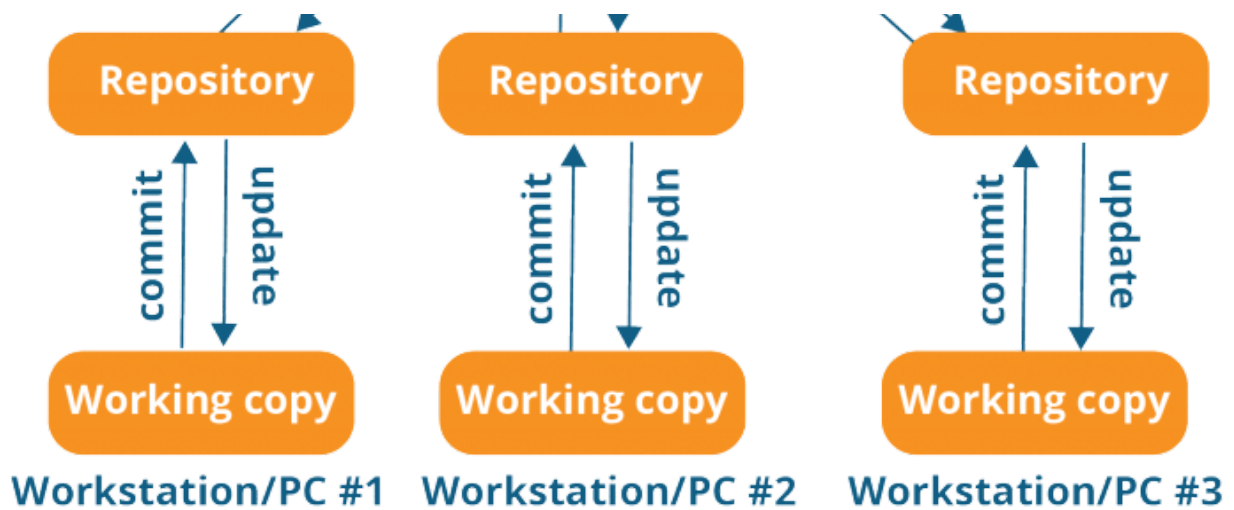
# calculate sum of
# digit
s=$(( $s + $k ))

done
echo "sum of digits of $g is : $s"
```

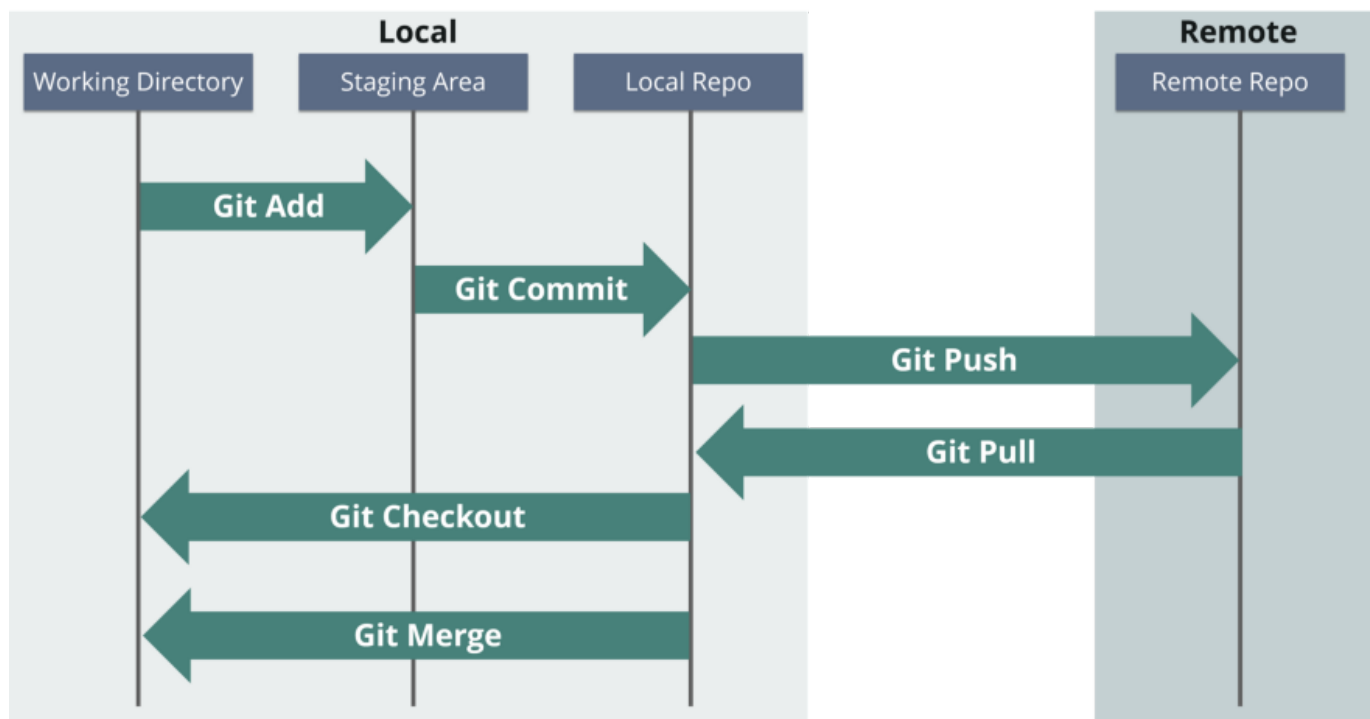
Git Commands

What is Git?





Git is a free, open-source distributed version control system. This tool handles everything from small to very large projects with speed and efficiency. Linus Torvalds created it in 2005 to develop the Linux Kernel. Git has the functionality, performance, security, and flexibility that most teams and individual developers need.



Tools like Git enable communication between the development and the operations team. When you are developing a large project with a huge number of collaborators, it is very important to have communication between the collaborators while making changes in the project. Commit messages in Git plays a very important role in communicating among the team. The bits and pieces that we all deploy lie in the Version Control system

like Git. To succeed in DevOps, you need to have all of the communication in Version Control. Hence, Git plays a vital role in succeeding at DevOps.

Git Commands

git init

Usage: `git init [repository name]`

This command creates a new repository.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo
$ git init
Initialized empty Git repository in C:/Users/Arvind Phulare/Desktop/edureka-repo/.git/
```

git config

Usage: `git config --global user.name "[name]"`

Usage: `git config --global user.email "[email address]"`

This command sets the author name and email address respectively. This is useful information with the commits.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git config --global user.name "arvind11"

Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git config --global user.email "arvind11@gmail.com"
```

git clone

Usage: `git clone [url]`

This command lets you get a copy of a repository from an existing URL.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git clone https://github.com/ArvindEd/Edureka1.git
Cloning into 'Edureka1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

git add

Usage: `git add [file]`

This command adds a file to the staging area.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git add file.txt
```

Usage: `git add *`

This command adds one or more to the staging area.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git add *
```

git commit

Usage: `git commit -m "[Type in the commit message]"`

This command records or snapshots the file permanently in the version history.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git commit -m "first commit"
[master (root-commit) c025eb2] first commit
2 files changed, 4 insertions(+)
create mode 160000 Edureka1
create mode 100644 file.txt
```

Usage: `git commit -a`

This command commits any files you've added with the git add command and also commits any files you've changed since then.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git commit -a
On branch master
nothing to commit, working tree clean
```

git status

Usage: `git status`

The git status command displays the state of the working directory and the staging area. This command lets you see the changes that are in the staging, those that are not staged and are not tracked by Git.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

git show

Usage: `git show [commit]`

This command shows the metadata and content changes of the specified commit.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git show
commit c025eb2ca95ab7f94cfcb27bbf2aa017fdb87e65 (HEAD -> master)
Author: arvind11 <arvind11@gmail.com>
Date: Tue Sep 17 15:14:39 2019 +0530

    first commit

diff --git a/Edureka1 b/Edureka1
new file mode 160000
index 0000000..ab318e6
--- /dev/null
+++ b/Edureka1
@@ -0,0 +1 @@
+Subproject commit ab318e6cbc7121d628d544748007ac1a2aff67c0
diff --git a/file.txt b/file.txt
new file mode 100644
index 0000000..d433f38
--- /dev/null
+++ b/file.txt
@@ -0,0 +1,3 @@
+
+
+Hi, How are you?
```

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)
$ git show f9b28577d0a8ee7cc1628d7ba2dd226605f1f308
commit f9b28577d0a8ee7cc1628d7ba2dd226605f1f308 (HEAD -> master)
```

```
Author: arvind11 <arvind11@gmail.com>  
Date: Tue Sep 17 15:21:36 2019 +0530
```

2nd file

```
diff --git a/file.py b/file.py  
new file mode 100644  
index 0000000..c40bc8f  
--- /dev/null  
+++ b/file.py  
@@ -0,0 +1,3 @@  
+  
+  
+print("Hello world")
```

git rm

Usage: `git rm [file]`

This command deletes the file from your working directory and stages the deletion.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)  
$ git rm file.py  
rm 'file.py'
```

git remote

Usage: `git remote add [variable name] [Remote Server Link]`

This command connects your local repository to the remote server.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo (master)  
$ git remote add origin https://github.com/ArvindEd/Edureka.git
```

git push

Usage: `git push [variable name] master`

This command sends the committed changes of the master branch to your remote repository.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (master)  
$ git push origin master  
Enumerating objects: 4, done.  
Counting objects: 100% (4/4), done.
```

```
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes | 97.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ArvindEd/Edureka1.git
ab318e6..bf23c2b master -> master
```

Usage: `git push [variable name] [branch]`

This command sends the branch commits to your remote repository.

Usage: `git push -all [variable name]`

This command pushes all branches to your remote repository.

Usage: `git push [variable name] :[branch name]`

This command deletes a branch on your remote repository.

git pull

Usage: `git pull [Repository Link]`

This command fetches and merges changes on the remote server to your working directory.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (master)
$ git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ArvindEd/Edureka1
 * branch            master      -> FETCH_HEAD
  bf23c2b..16e249f  master      -> origin/master
Updating bf23c2b..16e249f
Fast-forward
 pqr.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 pqr.py
```

git branch

Usage: `git branch`

This command lists all the local branches in the current repository.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (master)
$ git branch
* master
```

Usage: `git branch [branch name]`

This command creates a new branch.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (master)
$ git branch b1
```

Usage: `git branch -d [branch name]`

This command deletes the feature branch.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (master)
$ git branch -d b1
Deleted branch b1 (was 16e249f).
```

git checkout

Usage: `git checkout [branch name]`

This command lets you switch from one branch to another.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (master)
$ git checkout b2
Switched to branch 'b2'
```

Usage: `git checkout -b [branch name]`

This command creates a new branch and also switches to it.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (b2)
$ git checkout -b b3
Switched to a new branch 'b3'
```

git merge

Usage: `git merge [branch name]`

This command merges the specified branch's history into the current branch.

```
Arvind Phulare@DESKTOP-IEDC4TM MINGW64 ~/Desktop/edureka-repo/Edureka1 (b3)
$ git merge b3
Already up to date.
```

git rebase

Usage: `git rebase [branch name]`

git rebase master — This command will move all our work from the current branch to the master.

With this, We have come to the end of the article on Linux commands in DevOps. I have tried to cover as many commands as possible here. This article will definitely help you kick-start your journey with DevOps.

If you wish to check out more articles on the market's most trending technologies like Artificial Intelligence, Python, Ethical Hacking, then you can refer to [Edureka's official site](#).

Do look out for other articles in this series which will explain the various other aspects of DevOps.

1. [DevOps Tutorial](#)

2. [Git Tutorial](#)

3. [Jenkins Tutorial](#)

4. [Docker Tutorial](#)

5. [Ansible Tutorial](#)

6. [Puppet Tutorial](#)

7. [Chef Tutorial](#)

8. [Nagios Tutorial](#)
9. [How To Orchestrate DevOps Tools?](#)
10. [Continuous Delivery](#)
11. [Continuous Integration](#)
12. [Continuous Deployment](#)
13. [Continuous Delivery vs Continuous Deployment](#)
14. [CI CD Pipeline](#)
15. [Docker Compose](#)
16. [Docker Swarm](#)
17. [Docker Networking](#)
18. [Ansible Vault](#)
19. [Ansible Roles](#)
20. [Ansible for AWS](#)
21. [Jenkins Pipeline](#)
22. [Top Docker Commands](#)
23. [Git vs GitHub](#)
24. [Top Git Commands](#)
25. [DevOps Interview Questions](#)
26. [Who Is A DevOps Engineer?](#)
27. [DevOps Life cycle](#)

28. [Git Reflog](#)
 29. [Ansible Provisioning](#)
 30. [Top DevOps Skills That Organizations Are Looking For](#)
 30. [Waterfall vs Agile](#)
 31. [Jenkins CheatSheet](#)
 32. [Ansible Cheat Sheet](#)
 33. [Ansible Interview Questions And Answers](#)
 34. [50 Docker Interview Questions](#)
 35. [Agile Methodology](#)
 36. [Jenkins Interview Questions](#)
 37. [Git Interview Questions](#)
 38. [Docker Architecture](#)
 39. [Maven For Building Java Applications](#)
 40. [Jenkins vs Bamboo](#)
 41. [Nagios Tutorial](#)
 42. [Nagios Interview Questions](#)
 43. [DevOps Real-Time Scenarios](#)
 44. [Difference between Jenkins and Jenkins X](#)
 45. [Docker for Windows](#)
 46. [Git vs Github](#)
-

Originally published at <https://www.edureka.co> on September 19, 2019.

[Terminal](#)

[Linux](#)

[Linux Tutorial](#)

[DevOps](#)

[Linux Commands](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

