

Linux for Beginners: A Small Guide (Part 2)

December 4, 2020 By Raj Chandel

Let's dig in deeper from the previous concepts of [part 1 of this article](#) where we learned some basic day to day commands like navigating around the directories, creating files, copying them, commands to manipulate text on your terminal windows, installing software packages on your system and playing with file permissions.

In this article, we'll be focusing more on Linux fundamentals that provide us with a more granular level control over our operating system. Let's dig right into it.

Table of Content:

- Managing networks
- Process management
- Environment variables
- Conclusion

Managing Networks

Networking is a crucial topic for any aspiring penetration tester. A lot of times you would be required to test a network or something over it. Hence, it becomes important to know you to connect and interact with all of your network devices.

Let's get started with learning all the various tools and utilities to analyze and manage networks.

Ifconfig: Analyzing networks

The ifconfig command is one of the most basic tools for interacting with active network interfaces. Here we run ifconfig and we can see the IP address mapped to our 2 network interfaces: **eth0** and **lo**.

We can also see the **netmask** and a **broadcast address** of the network interface attached. As well as the **mac address** which I have blurred out.

(lo is localhost and is always mapped to 127.0.0.1)

```

[root@Kali]~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.51 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::20c:29ff:fe86:9ae0 prefixlen 64 scopeid 0x20<link>
    RX packets 15 bytes 1900 (1.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 15 bytes 1356 (1.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19 base 0x2000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 16 bytes 796 (796.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 16 bytes 796 (796.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

iwconfig: Checking wireless network devices

If you have a wireless adapter, you can use the iwconfig command to gather crucial information such as its IP address, MAC address, which mode it is in and much more. Since I don't have a wireless adapter, my output is as such.

```

[root@Kali]~# iwconfig
lo          no wireless extensions.

eth0        no wireless extensions.

```

Changing your IP Address

In order to change your IP address, enter ifconfig, the interface you want to change the address for and the new address you want to assign to it. Let's change the IP address to **192.168.1.13**.

Upon running ifconfig we see the change reflected.

```
ifconfig eth0 192.168.1.13
```

```

[root@Kali]~# ifconfig eth0 192.168.1.13
[root@Kali]~# ifconfig | grep inet
    inet 192.168.1.13 netmask 255.255.255.0
    inet6 fe80::20c:29ff:fe86:9ae0 prefixlen 64 scopeid 0x20<link>
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>

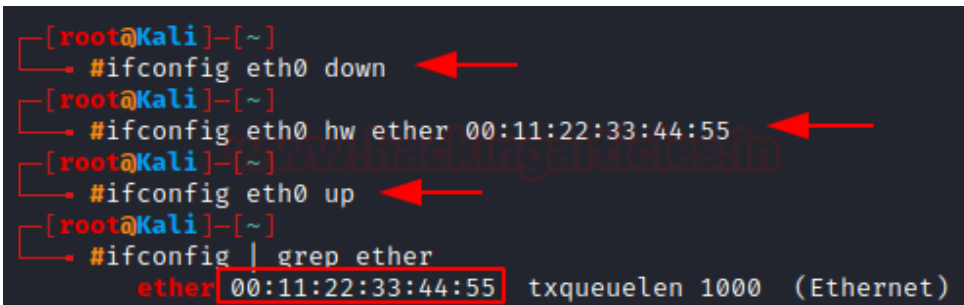
```

Spoofing your MAC Address

You can also use `ifconfig` to change your MAC address. Since MAC address is globally unique and it often used as a security measure to keep the hackers out of networks or even to trace them, spoofing your MAC address is almost trivial in order to neutralize these security measures and maintain anonymity.

In order to change our MAC address to `00:11:22:33:44:55`, we'll have to down the interface, change the MAC address and then up the interface again.

```
ifconfig eth0 down
ifconfig eth0 hw ether 00:11:22:33:44:55
ifconfig eth0 up
```

A terminal window showing the process of changing the MAC address of the eth0 interface. The commands entered are: `#ifconfig eth0 down`, `#ifconfig eth0 hw ether 00:11:22:33:44:55`, and `#ifconfig eth0 up`. Red arrows point to each of these three commands. The final command shown is `#ifconfig | grep ether`, which outputs `ether 00:11:22:33:44:55 txqueuelen 1000 (Ethernet)`. The MAC address `00:11:22:33:44:55` is highlighted with a red box.

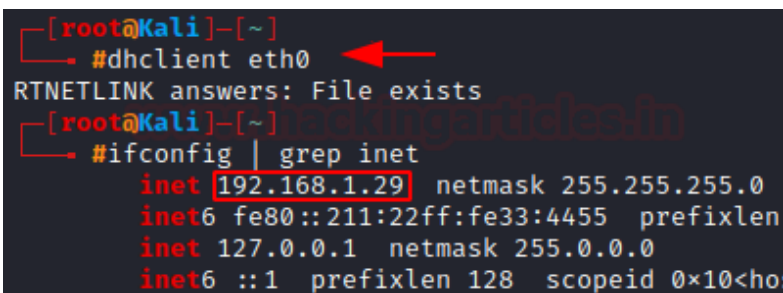
Using DHCP Server to assign new IP Addresses

Linux has a Dynamic Host Configuration Protocol (DHCP) server that runs a daemon – a process that runs in the background called DHCP daemon. This DHCP server assigns IP addresses to all the systems on the subnet and it also keeps log files of such.

Let's request an IP Address from DHCP, by simply calling the DHCP server with the command **dhclient** and **network interface** you would want to change the IP Address of.

We can see the IP Address has changed from what we had manually given it earlier.

```
dhclient eth0
```

A terminal window showing the process of requesting an IP address from a DHCP server. The command entered is `#dhclient eth0`, with a red arrow pointing to it. The output is `RTNETLINK answers: File exists`. The next command is `#ifconfig | grep inet`, which outputs several lines of network configuration. The first line is `inet 192.168.1.29 netmask 255.255.255.0`, where the IP address `192.168.1.29` is highlighted with a red box. Other lines show `inet6` configurations.

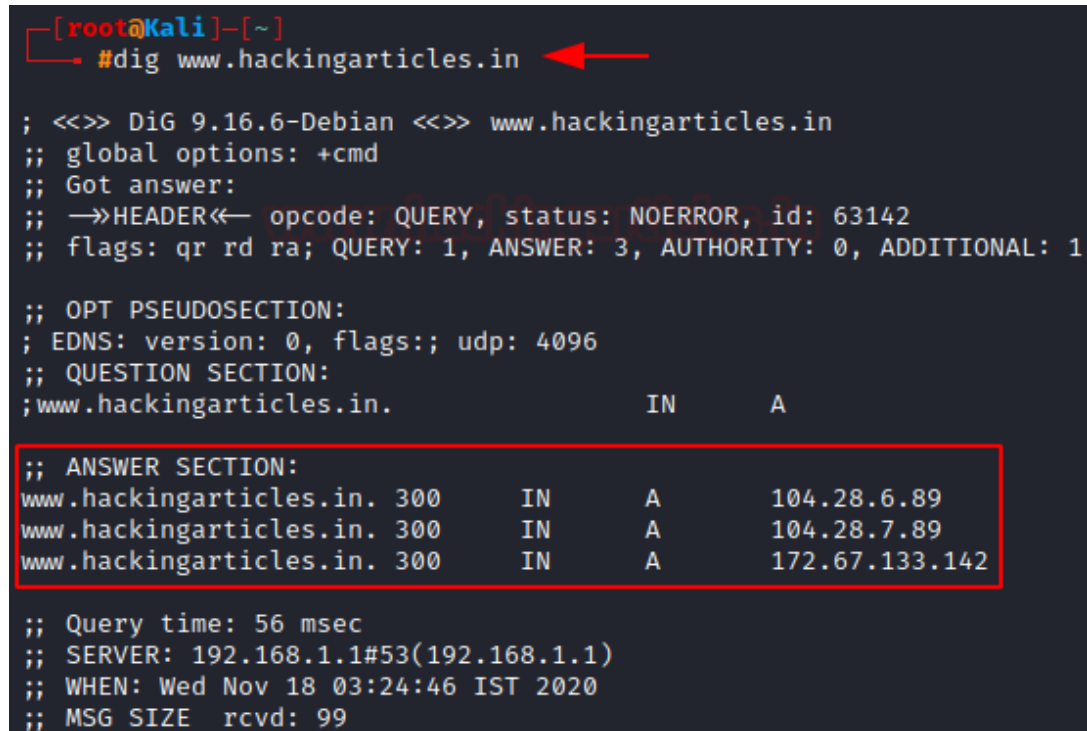
Examining DNS with dig

DNS is a service that translates a domain name like “**hackingarticles.in**” to the appropriate IP address. We can use the dig command with added options such as mx (mail server), ns (name sever) to gather more information regarding the domain and its mail and name servers respectively.

Let’s use the dig command on “**www.hackingarticles.in**” here we can see the domain name resolve into IP Address.

```
dig www.hackingarticles.in
```

```
[root@Kali]~# dig www.hackingarticles.in
```



```
; <<>> DiG 9.16.6-Debian <<>> www.hackingarticles.in
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 63142
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.hackingarticles.in.                IN      A

;; ANSWER SECTION:
www.hackingarticles.in. 300      IN      A      104.28.6.89
www.hackingarticles.in. 300      IN      A      104.28.7.89
www.hackingarticles.in. 300      IN      A      172.67.133.142

;; Query time: 56 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Nov 18 03:24:46 IST 2020
;; MSG SIZE rcvd: 99
```

Further searching “**hackingarticles.in**” mail servers:

```
dig hackingarticles.in mx
```

```

[root@Kali]~#dig hackingarticles.in mx
; <<>> DiG 9.16.6-Debian <<>> hackingarticles.in mx
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 5602
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;hackingarticles.in.      IN      MX

;; ANSWER SECTION:
hackingarticles.in.      300     IN      MX      20 alt1.aspmx.l.google.com.
hackingarticles.in.      300     IN      MX      10 aspmx.l.google.com.
hackingarticles.in.      300     IN      MX      30 aspmx2.googlemail.com.

;; Query time: 24 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Nov 18 03:25:42 IST 2020
;; MSG SIZE rcvd: 136

```

Searching for the name servers:

```
dig hackingarticles.in ns
```

```

[root@Kali]~#dig hackingarticles.in ns
; <<>> DiG 9.16.6-Debian <<>> hackingarticles.in ns
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 59582
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;hackingarticles.in.      IN      NS

;; ANSWER SECTION:
hackingarticles.in.      86400   IN      NS      duke.ns.cloudflare.com.
hackingarticles.in.      86400   IN      NS      kay.ns.cloudflare.com.

;; Query time: 60 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Nov 18 03:26:47 IST 2020
;; MSG SIZE rcvd: 101

```

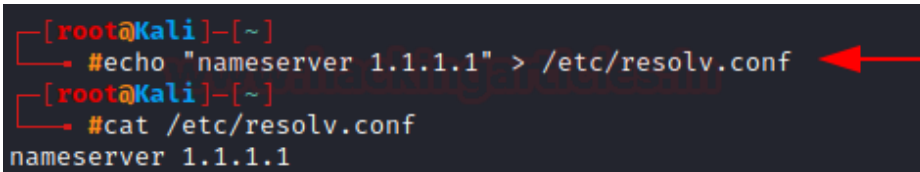
Changing your DNS Server

The DNS server information is stored in `/etc/resolv.conf`, in order to change the DNS server we need to edit this file. We can simply use **nano** or **vim** which are some of the common text editors Linux.

Here, we are going to use the **echo** command and **>** to overwrite the **resolve.conf** file. We can see the change reflect when reading using **cat**.

- is Cloudflare's public DNS server, you could also use Google's which is 8.8.8.8)

```
echo "nameserver 1.1.1.1" > /etc/resolv.conf
```



```
[root@Kali]~# echo "nameserver 1.1.1.1" > /etc/resolv.conf  
[root@Kali]~# cat /etc/resolv.conf  
nameserver 1.1.1.1
```

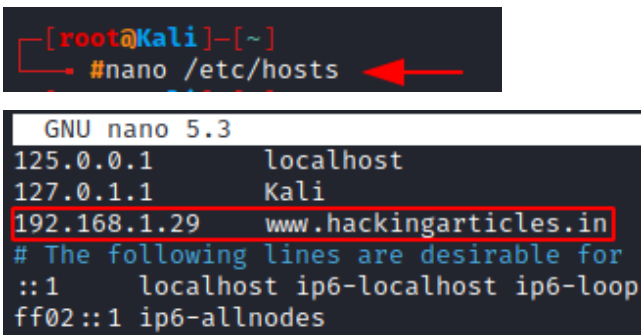
A red arrow points to the command being executed in the terminal.

Mapping the IP Addresses

There is a file in our system called **hosts** which also performs domain name – IP Address translation. The file is located in **/etc/hosts**. We can map any domain to the IP address of our choice, this can be useful as the hacker to direct traffic from network to a malicious web server (using dnspooft).

Let's nano into the file. Here we can see localhost and kali mapped to certain IP addresses. We can map **www.hackingarticles.in** to our IP address. Now if anyone on the network goes to this URL it will be re-directed to our IP address, we can further run an apache server and deploy a malicious website, tricking the users in the network.

```
nano /etc/hosts
```



```
[root@Kali]~# nano /etc/hosts  
GNU nano 5.3  
125.0.0.1    localhost  
127.0.1.1    Kali  
192.168.1.29 www.hackingarticles.in  
# The following lines are desirable for  
::1         localhost ip6-localhost ip6-loop  
ff02::1     ip6-allnodes
```

A red arrow points to the command being executed in the terminal. The line **192.168.1.29 www.hackingarticles.in** is highlighted with a red box in the nano editor.

Process Management

A process is just a program that's running on your system and consuming resources. There are times when a particular process has to be killed because it's malfunctioning or as a pen-tester, you would want to stop the anti-virus applications or firewalls. We'll learn how to discover and manage such processes in this section.

Viewing process

In order to manage the process, we must be able to view them first. The primary tool to do so is **ps**.

Simple typing **ps** in the bash shell will list down all the **active processes**.

(PID stands for process ID and is unique for every invoked process.)

```
[root@Kali]~# ps
  PID TTY          TIME CMD
 4832 pts/0    00:00:00 sudo
 4833 pts/0    00:00:00 su
 4834 pts/0    00:00:00 bash
 6117 pts/0    00:00:00 ps
```

Viewing process for all the users

Running **ps** command with **aux**, will display **all the running processes for all users**, so let's run

```
ps aux
```

Here we can see PID, the user who invoked the process, %CPU the process is using, %MEM represent the percentage of memory being used and finally COMMAND which is the name of the command that has started the process

```
[root@Kali]~# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.2 168596 11860 ?        Ss   03:13   0:01 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    03:13   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   03:13   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   03:13   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   03:13   0:00 [kworker/0:0H-kblockd]
```

Filtering Process with its name

As we learned earlier, we can pipe the output of **ps aux** into **grep** and filter out the specific information we want.

Let's search for **msfconsole** (A popular interface to use the Metasploit framework)

```
ps aux | grep msfconsole
```

```
[root@Kali]~# ps aux | grep msfconsole
root        6152  0.0  0.0  6112   644 pts/0    S+   03:53   0:00 grep --color=auto msfconsole
```

top: Finding the greediest process

In some use cases when you want to know which process is using the most resources, we use the **top** command. It displays the process ordered by the resources used. Unlike ps, the top also refreshed dynamically – every 10

seconds.

```
[root@Kali]~#top
```

```
top - 03:53:54 up 40 min, 1 user, load average: 0.14, 0.18, 0.18
Tasks: 261 total, 2 running, 259 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.3 sy, 0.0 ni, 99.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3938.0 total, 2014.1 free, 1077.5 used, 846.4 buff/cache
MiB Swap: 6675.0 total, 6675.0 free, 0.0 used. 2579.0 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4098	root	20	0	428268	138496	68452	R	2.3	3.4	0:57.85	Xorg
4567	karan	20	0	514704	39116	31664	S	0.7	1.0	0:01.93	panel-17-pulsea
1364	root	20	0	164036	9660	6576	S	0.3	0.2	0:04.38	vmtoolsd
4332	karan	20	0	275100	25696	17488	S	0.3	0.6	0:00.49	xfce4-session
4411	karan	20	0	400316	76704	58304	S	0.3	1.9	0:06.95	xfwm4
4553	karan	20	0	450148	117944	49180	S	0.3	2.9	0:03.02	xfdesktop

Changing Priority with the “nice” command

When you start a process, you can set its priority level with the **nice** command. Let's increment the priority of **/usr/bin/ssh-agent** by **10** (increasing its priority) using the **n** tag.

```
nice -n -10 /usr/bin/ssh-agent
```

```
[root@Kali]~#nice -n -10 /usr/bin/ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-ePY6JdX08FUY/agent.6241; export SSH_AUTH_SOCK;
SSH_AGENT_PID=6242; export SSH_AGENT_PID;
echo Agent pid 6242;
```

The “renice” Command

The **renice** command takes an absolute value between -20 and 19 and sets the priority to that particular level. It also required the PID (process ID).

Let's give a process of **PID 6242** a higher level of priority. (increment it by 20)

```
renice 20 6242
```

```
[root@Kali]~#renice 20 6242
6242 (process ID) old priority -10, new priority 19
```

kill : The deadliest Command

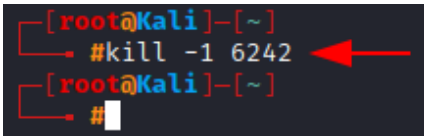
At times, when a process exhibits unusual behaviour or consumes too many system resources, they are called a **zombie process**. In order to stop these kinds of processes, we use the **kill** command.

The **kill** command has 64 different kill signals, each signifying something slightly different.

(1 stands for Hangup and is designated to stop the process while 9 is the absolute kill, it forces the process to stop by sending its resources to /dev/null).

Let's stop the process 6242

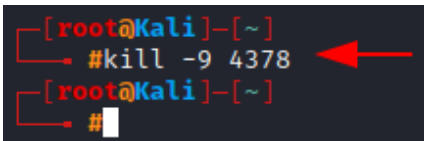
```
kill -1 6242
```



```
[root@Kali]~#  
#kill -1 6242  
[root@Kali]~#
```

And in order to force stop process 4378

```
kill -9 4378
```



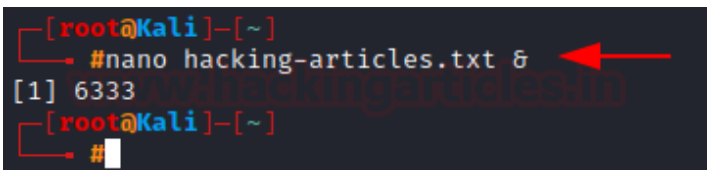
```
[root@Kali]~#  
#kill -9 4378  
[root@Kali]~#
```

Running processes in the background

At times, you may want a process to run in the background, and we can do so by simply adding **&** to the end of the command.=

Let's run **nano** in the background. (You can see the PID that is generated)

```
nano hacking-articles.txt &
```

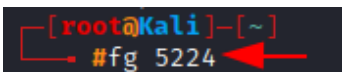


```
[root@Kali]~#  
#nano hacking-articles.txt &  
[1] 6333  
[root@Kali]~#
```

Moving a process to the foreground

If you want to move a process running in the background to the foreground, you can use the **fg** command. Simply type **fg** and then the **process ID**.

(In order to see the background processes in your system simply use the command **jobs**)



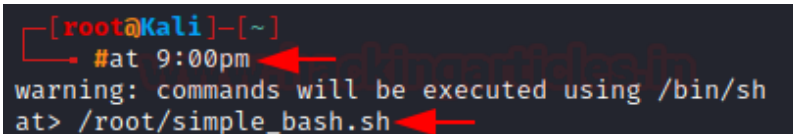
```
[root@Kali]~#  
#fg 5224
```

Scheduling a process

Often one might need to schedule processes to run at a particular time of day. The **at** command is a daemon – a background process which is useful for scheduling a job to run once at some point in the future. While for jobs that occur every day, week, the **crond** is more suited.

Let's execute a **scanning_script.sh** at **9:30pm**.

```
at 9:00pm
/root/simple_bash.sh
```

A terminal window with a dark background. The prompt is [root@Kali]~. The user enters #at 9:00pm. A warning message appears: warning: commands will be executed using /bin/sh. The user then enters at> /root/simple_bash.sh. Red arrows point to the at command and the file path.

```
[root@Kali]~
#at 9:00pm
warning: commands will be executed using /bin/sh
at> /root/simple_bash.sh
```

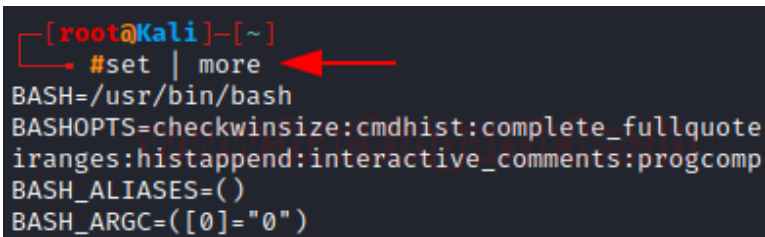
User Environment Variables

Understanding environment variables is a must when trying to get the most from your Linux system, it is crucial to be able to manage them for optimal performance. Variables are just strings in key-value pairs. There are two types of variables, environment and shell, while the shell variables are only valid for the particular session, the environment variables are system-wide.

Viewing all the Environment Variables

You can view all your default environment variables by entering **env** into your terminal from any directory, like so:

```
set | more
```

A terminal window with a dark background. The prompt is [root@Kali]~. The user enters #set | more. The output shows various environment variables: BASH=/usr/bin/bash, BASHOPTS=checkwinsize:cmdhist:complete_fullquote, iranges:histappend:interactive_comments:progcomp, BASH_ALIASES=(), and BASH_ARGC=([0]="0"). A red arrow points to the command.

```
[root@Kali]~
#set | more
BASH=/usr/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote
iranges:histappend:interactive_comments:progcomp
BASH_ALIASES=()
BASH_ARGC=([0]="0")
```

Filtering for particular variables

Again, using **piping** the output to the **grep** command we can filter out the variables we want.

Let's filter out **HISTSIZE** (history size)

As we can see the history size is set to 1000.

```
set | grep HISTSIZE
```

```
[root@Kali]~  
#set | grep HISTSIZE  
HISTSIZE=1000  
[root@Kali]~  
#
```

Changing variable value temporarily

We can change the variable values simply by typing out the variable and equating it to a new value but this new value will only be changed for this particular session, if you open a new terminal window it will change back to its default.

After running this, you'll see that when you press the up/down arrow keys to recall your previous commands, nothing happens since we changed to a number of commands being stored to 0.

```
HISTSIZE = 0
```

```
[root@Kali]~  
#HISTSIZE=0  
[root@Kali]~  
#
```

Making the changes permanent

When changing the variables, it is always best practice to store the default value in say, a text. This way you can always undo your changes.

Let's **echo** the value into a text file name **valueofHISTSIZE** and save it in our working directory by adding ~/

```
echo $HISTSIZE ~/valueofHISTSIZE.txt
```

```
[root@Kali]~  
#echo $HISTSIZE ~/valueofHISTSIZE.txt  
1000 /root/valueofHISTSIZE.txt
```

Now, just like last time change the value of **HISTSIZE** but now we'll execute another command **export**. Which will make this change permanent.

```
HISTSIZE=0  
export HISTSIZE
```

```
[root@Kali]~  
#HISTSIZE=0  
[root@Kali]~  
#export HISTSIZE
```

Creating user-defined variables

You can also design your custom, user-defined variables just by assigning a value to a new value name of your choice.

Let's create a new variable called **URL** which has the value **www.hackingarticles.in**.

```
url_variable="www.hackingarticles.in"
```

```
[root@Kali]~  
#url_variable="www.hackingarticles.in"  
[root@Kali]~  
#echo $url_variable  
www.hackingarticles.in
```

We can also delete this variable by using the **unset** command. Simply typing unset and the name of the variable will do the trick.

As we can see, there is no result despite running the **echo** command.

```
unset url_variable
```

```
[root@Kali]~  
#unset url_variable  
[root@Kali]~  
#echo $url_variable
```

Conclusion

The topics we learned here; help you understand a bit more of the inner workings of Linux. Practicing and applying these concepts as well as digging in deeper yourself is the way to go now. Be sure to be on the lookout for the **3rd part** of this article, where we discuss more advanced Linux concepts like **Bash Scripting**, **automation** and using **Linux services**.