

Jesua Gonzalez

Ethical Hacking

Linux Fundamentals & System Administration

Thu Sep 18

Through these interactive labs, I have learned fundamental Linux skills that are key to my success in penetration testing and overall ethical hacking. The Labtainer exercises helped me develop core competencies in user administration, file system management, and command-line navigation. These skills form the groundwork for diving into the more complex world of cybersecurity and future technical techniques I may encounter. The first lab introduced me to the most commonly used commands to navigate around directories and files. In essence, I was able to locate text files or patterns using commands like grep alongside commands such as find. I was also exposed to ways of creating/deleting new directories or files while simultaneously applying the correct permissions where needed, using commands such as chmod. In addition, I was able to understand the way in which a script is created using Leafpad, one of many editors, and then executing the new script. The demonstration used was pinging a known web server like Google.com to confirm my network IP was up and running.

In the second lab, I was able to further comprehend the foundation of user access management. To further explain, I was able to create 3 different users who each had a certain purpose and task. The first user, Bob, was added using useradd while I was in root admin. I was able to then create Bob's password, but also making sure it expires after he attempts to log in; this was done through the passwd program. Besides Bob, Mary was also created in order to be added to the Bakers group to access key files for her to conduct her duties. Lastly, Lisa's user account was made the same way as Bob's and Mary's, but she was granted privileged access. Lisa was added using usermod just as Mary, and then was able to remove/delete users no longer needed. Overall, these labs gave me a greater understanding of linux operations.

## Lab 1: Nix-Commands

```
student@nix-commands:~$ pwd
/home/student
student@nix-commands:~$ ls
student@nix-commands:~$ ls -a
. .bash_history .bashrc .profile
.. .bash_logout .local .sudo_as_admin_successful
student@nix-commands:~$ la -al
total 40
drwxr-xr-x 1 student student 4096 Sep 19 01:42 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw-r--r-- 1 student student 20 Sep 19 01:46 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 .sudo_as_admin_successful
student@nix-commands:~$ ll
total 40
drwxr-xr-x 1 student student 4096 Sep 19 01:42 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw-r--r-- 1 student student 56 Sep 19 01:52 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 .sudo_as_admin_successful
student@nix-commands:~$
```

**Linux Tutorial  
nix-commands.pdf**

In addition to the files you can see, there are other files that are "hidden". In Unix, hidden files are files that you don't necessarily want to see all the time. Therefore, unless specifically requested, they are not shown with a directory listing. Any file or directory that begins with a '.' is hidden, and is referred to as a *dot file*.

**ls -a**

List all objects, even the hidden files and directories, by also using the **-a** (all) option, as shown below. (Note that there must always be a space between the command and any arguments being passed to it).

**ls**

The home directory contains many user-level configuration files for modifying how your environment behaves, and they are almost always dot files or dot directories. The ".login" file (if it exists) controls things you always want done when you log in.

**cd temp**

Change your current directory to the new directory using the **cd** (change directory) command:

**pwd**

List all the contents of this new directory:

**ll -a**

3 Rev: 2021-7-8

- Starting this lab, we see that we'll be working with the command line interface. First, 'pwd' is used to print out our current directory. 'ls' is then used to display any files and content in the current directory. Similarly, 'ls -a' lists hidden text files that start with a '.' and 'll' will be able to display their permissions and file details.

```
student@nix-commands:~$ mkdir temp
student@nix-commands:~$ ll
total 44
drwxr-xr-x 1 student student 4096 Sep 19 01:52 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw-r--r-- 1 student student 56 Sep 19 01:52 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 .sudo_as_admin_successful
drwxrwxr-x 2 student student 4096 Sep 19 01:52 temp/
student@nix-commands:~$ cd temp
student@nix-commands:~/temp$ pwd
/home/student/temp
student@nix-commands:~/temp$ ll -a
total 12
drwxrwxr-x 2 student student 4096 Sep 19 01:52 .
drwxr-xr-x 1 student student 4096 Sep 19 01:52 ..
student@nix-commands:~/temp$ ll ..
total 44
drwxr-xr-x 1 student student 4096 Sep 19 01:52 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw-r--r-- 1 student student 88 Sep 19 01:54 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
```

**Linux Tutorial  
nix-commands.pdf**

Even though a directory is brand new, it is not exactly empty. Every directory has at least two entries: two directories named "." and ".." (called *dot* and *dotdot*). The *dotdot* directory is a shortcut for the current directory, while *dotdot* is a shortcut to the parent directory. Windows has borrowed this philosophy too. (One thing hackers do to hide their own files is to create a directory with three dots, which might be easily overlooked).

Try to delete the temp2 directory using the **rmdir** (remove directory) command:

**rmdir temp2**

It should have failed because files still exist in that directory. Delete the files in the temp2 directory by using the **rm** (remove) command:

**rm temp2/\***

Notice the use of the wild card "\*" symbol in the **rm** command. The command was interpreted to mean: delete all the files starting with ".bas".

Now remove the directory:

**rmdir temp2**

(Note that a **rmdir** will fail if you are

- Here we use 'mkdir', a command to create a new directory labeled temp. I am then able to enter that directory using the 'cd' command, short for change directory. And as before, 'll -a' helps display all files and their details, but combining 'll' with '..' will display content and details of the outside or parent directory before we entered "temp".

The screenshot shows a Linux desktop with a terminal window open. The terminal window title is "student@nix-commands: ~". The command history shows:

```

student@nix-commands:~/temp$ ll ..
total 44
drwxr-xr-x 1 student student 4096 Sep 19 01:52 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw----- 1 student student 80 Sep 19 01:54 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 .sudo_as_admin_successful
drwxrwxr-x 2 student student 4096 Sep 19 01:52 temp/
student@nix-commands:~/temp$ cd ..
student@nix-commands:~$ pwd
/home/student
student@nix-commands:~$ mv temp temp2
student@nix-commands:~$ ll
total 44
drwxr-xr-x 1 student student 4096 Sep 19 01:56 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw----- 1 student student 107 Sep 19 01:56 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 .sudo_as_admin_successful
drwxrwxr-x 2 student student 4096 Sep 19 01:52 temp2/
student@nix-commands:~$ 

```

To the right of the terminal is a PDF titled "Linux Tutorial nix-commands.pdf" which contains a section on file operations. The text in the PDF includes:

- Use the dotdot directory to change your working directory to the parent directory of temp, as shown below:
 

```
11 ..
```
- Use the dotdot directory to list the contents of the parent directory:
 

```
11 ..
```
- Notice that a nice feature of the cd command is that if it is entered with no arguments, it will always take you back to your home directory.
- Files or directories can be moved or simply renamed by using the mv (move) command. Rename the temp directory to temp2 by doing the following:
 

```
mv temp temp2
```
- Copying files is done with the cp (copy) command. Copy one of your hidden files into the temp2 directory:
 

```
cp .bashrc temp2
```
- The echo command may seem meaningless, but it comes in handy, such as:
 

```
echo "hello world"
```

- Note that ‘..’, otherwise known as dotdot, can also be used with ‘cd’ to change to the parent directory without specifically stating the path to it. I then use the ‘mv’ command to rename temp to temp2. This command is also able to move files or directories to other locations besides just renaming them.

The screenshot shows a Linux desktop with a terminal window open. The terminal window title is "student@nix-commands: ~". The command history shows:

```

student@nix-commands:~$ cp .bashrc temp2
cp: cannot stat '.bashrc': No such file or directory
student@nix-commands:~$ ll -a temp2
total 16
drwxrwxr-x 2 student student 4096 Sep 19 01:57 .
drwxr-xr-x 1 student student 4096 Sep 19 01:56 ..
-rw-r--r-- 1 student student 3921 Sep 19 01:57 .bashrc
student@nix-commands:~$ cp .bashrc temp2/.bashrc
student@nix-commands:~$ ll -a temp2
total 20
drwxrwxr-x 2 student student 4096 Sep 19 01:59 .
drwxr-xr-x 1 student student 4096 Sep 19 01:56 ..
-rw-r--r-- 1 student student 3921 Sep 19 01:59 .bashrc
-rw-r--r-- 1 student student 3921 Sep 19 01:57 .bashrc
student@nix-commands:~$ rmdir temp2
rmdir: failed to remove 'temp2': Directory not empty
student@nix-commands:~$ rm temp2/*.bas*
student@nix-commands:~$ ll -a temp2
total 12
drwxrwxr-x 2 student student 4096 Sep 19 02:01 .
drwxr-xr-x 1 student student 4096 Sep 19 01:56 ..
student@nix-commands:~$ rmdir temp2
student@nix-commands:~$ ll
total 40
drwxr-xr-x 1 student student 4096 Sep 19 02:01 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw----- 1 student student 243 Sep 19 02:01 .bash_history

```

To the right of the terminal is a PDF titled "Linux Tutorial nix-commands.pdf" which contains a section on file operations. The text in the PDF includes:

- Use the dotdot directory to change your working directory to the parent directory of temp, as shown below:
 

```
11 ..
```
- Use the dotdot directory to list the contents of the parent directory:
 

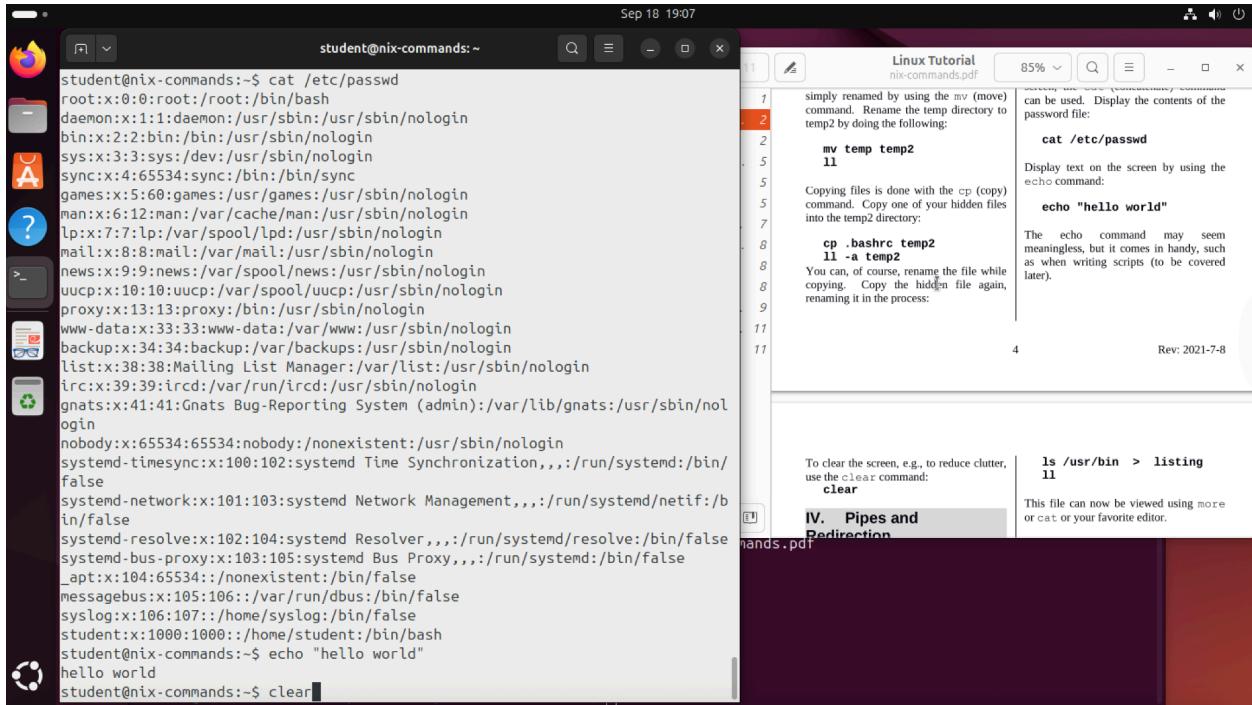
```
11 ..
```
- Notice that a nice feature of the cd command is that if it is entered with no arguments, it will always take you back to your home directory.
- Files or directories can be moved or simply renamed by using the mv (move) command. Rename the temp directory to temp2 by doing the following:
 

```
mv temp temp2
```
- Copying files is done with the cp (copy) command. Copy one of your hidden files into the temp2 directory:
 

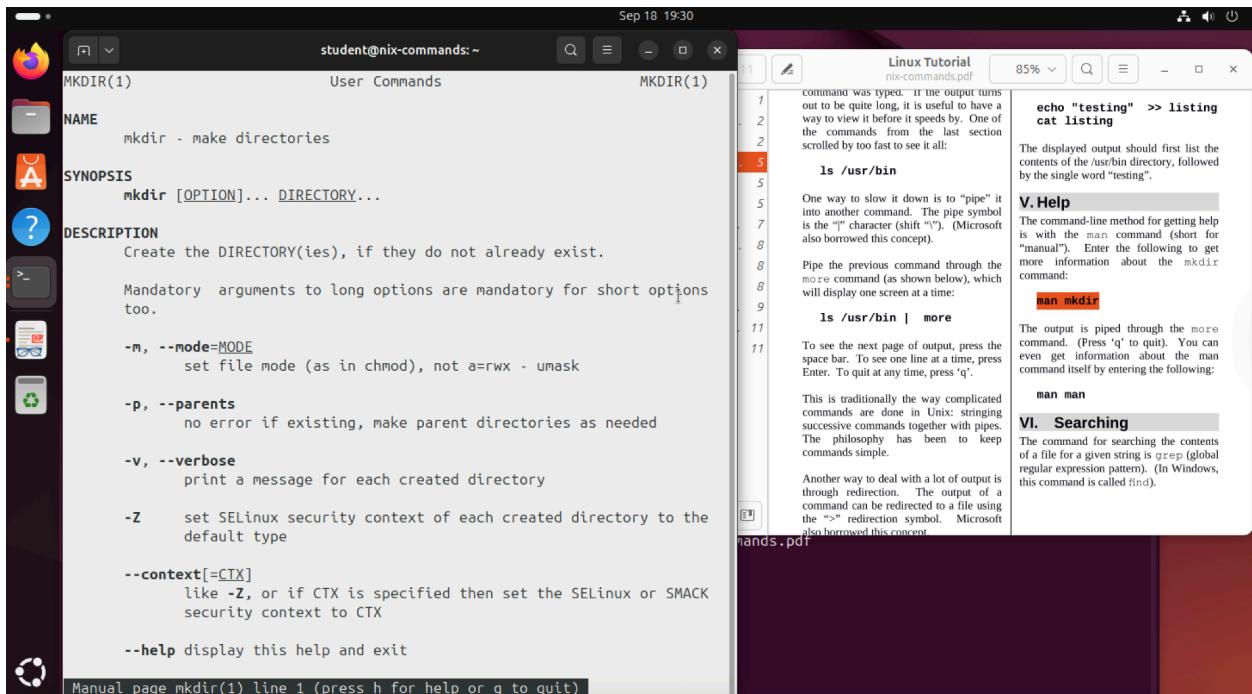
```
cp .bashrc temp2
```
- The echo command may seem meaningless, but it comes in handy, such as:
 

```
echo "hello world"
```

- I was able to copy a file named .bashrc into my temp2 directory using the ‘cp’ command. But there is also a way to copy a file into a directory using ‘cp’ as before, and then when specifying the location, you can add / and rewrite the file name in this case .bashrc to just .bash. I then explore the ‘rm’ command to remove all files starting with .bash using the ‘\*’ wildcard. We remove files first because a directory can't be removed if not empty.



- Here I can read and display file contents using the ‘cat’ command. As well as being able to display writing on the terminal using the ‘echo’ command. In order to have a clean terminal, we can simply write ‘clear’ and the display gets cleared.



- The ‘man’ command is a very useful tool that helps look in the manual for help in understanding a command or its associated attributes. It’s sort of like a dictionary for commands.

student@nix-commands:~\$ grep -s student /etc/\*
/etc/group:student:x:27:student
/etc/group:student:x:1000:
/etc/passwd:student:x:1000:1000::/home/student:/bin/bash
/etc/subgid:student:100000:65536
/etc/subuid:student:100000:65536
student@nix-commands:~\$ find /etc -name hosts -print
/etc/hosts
find: '/etc/ssl/private': Permission denied
student@nix-commands:~\$ sudo su
root@nix-commands:/home/student# find /etc -name hosts -print
root@nix-commands:/home/student# find /usr/include -name "\*.h" -print
/usr/include/netash/ash.h
/usr/include/tar.h
/usr/include/monetary.h
/usr/include/sqty.h
/usr/include/video/edid.h
/usr/include/video/sifb.h
/usr/include/video/uvesafb.h
/usr/include/uchar.h
/usr/include/printf.h
/usr/include/complex.h
/usr/include/sound/firewire.h
/usr/include/sound/hdspm.h

**Linux Tutorial  
nix-commands.pdf**

The Unix `find` command is like the Swiss army knife of Unix commands, and is also very useful for finding things. Its syntax is somewhat complicated, and it can do much more than can be shown in this tutorial.

One basic use of `find` is to locate a file with a known name. Use `find` to locate a file called "hosts", using the following command:

```
find /etc -name hosts -print
```

find recursively checks all the directories from the starting point down. In the above example, it looked everywhere at and below the "/etc" directory. The "-name hosts" tells `find` to search for all files/directories whose name is "hosts". The "print" tells `find` what to do when it finds the requested file(s). In this case, it prints out the path where it is located.

`find /usr/include -name *.h -print`

An even more basic use of `find` is to display the path of every file in trees. In the following example, `find` is told to look in the entire hierarchy starting with "/usr/local". When it finds a file, it prints out the location. In other words, it will display all the file and directory names in the hierarchy.

```
find /usr/local -print
```

Return to the privilege of a regular user:

```
exit
```

6 Rev: 2021-7-8

- The 'grep' command allows searching all files in reference to a certain keyword or text pattern. On the other hand, the 'find' command searches through subdirectories for a filename and file attributes.

student@nix-commands:~\$ exit
student@nix-commands:~\$ cd
student@nix-commands:~\$ pwd
/home/student
student@nix-commands:~\$ ll -a
total 48
drwxr-xr-x 1 student student 4096 Sep 19 02:12 .
drwxr-xr-x 1 root root 4096 Jul 7 2018 ..
-rw----- 1 student student 497 Sep 19 03:07 .bash\_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash\_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 .sudo\_as\_admin\_successful
-rw-rw-r-- 1 student student 5410 Sep 19 02:13 listing
student@nix-commands:~\$ ll .bashrc
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
student@nix-commands:~\$ chmod o+w .bashrc
student@nix-commands:~\$ ll .bashrc
-rw-r--rw- 1 student student 3921 Sep 8 00:37 .bashrc
student@nix-commands:~\$ chmod o-w .bashrc
student@nix-commands:~\$ ll .bashrc
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc
student@nix-commands:~\$ chmod ugo+rwx .bashrc
student@nix-commands:~\$ ll .bashrc
-rwxrwxrwx 1 student student 3921 Sep 8 00:37 .bashrc\*
student@nix-commands:~\$ chmod go=r .bashrc
student@nix-commands:~\$ ll .bashrc
-rwxr--r-- 1 student student 3921 Sep 8 00:37 .bashrc\*
student@nix-commands:~\$

**Linux Tutorial  
nix-commands.pdf**

else). These three entities (referred to as user, group and other) may be given one or more of the following permissions to files or directories: read, write, and execute.

**cd ll -a**

Display the contents of your home directory:

The permissions are displayed on the far left-hand side of the output of each object. The permissions are broken down by user (i.e., owner), group and others, as follows:

User	Group	Other
(r)	(w)	(x)

Each section is divided into one of three permission classes: read (r), write (w), and execute (x). If a permission has been granted then it appears, otherwise a '-' indicates the permission is not granted. In the above example, the user (or file owner) has all three permissions, while the group has read and write permissions, and other has only read permission.

**chmod ug+rw .bashrc**

read, write and execute, use 'r', 'w' and 'x', respectively. For example, change the permissions on your .bashrc file so that everyone can write to it:

**chmod o+w .bashrc**

The "o+w" means "Add the write permission to other". Now remove the permission:

**chmod o-w .bashrc**

Multiple changes can be made at one time. Do the following to make multiple changes at once:

**chmod ug+rwx .bashrc**

where user, group and other are all given read, write and execute permissions to the file.

To change the permissions so the group and other only have read access, do the following:

**chmod go=r .bashrc**

- I explored file permissions in this section, recalling that 'll -a' displays file details and all files/directories, even hidden ones. Permissions are grouped by 3 classes: User, Group, and Other. The first character in the permissions detail is either a 'd' for directory or blank '-' for a file type. 'rwx' or read, write, execute are the permissions that can be given per group, but of course can be changed. 'chmod' is the command used for changing perms followed by the class initial and add or subtract symbol and the perm.

The screenshot shows a Linux desktop environment. On the left, a terminal window titled "student@nix-commands:" displays the output of a ping command to google.com, showing 0% packet loss. On the right, a PDF viewer titled "Linux Tutorial" is open, displaying a page about the ping command. The page includes code examples for pinging multiple sites and creating a script to do so.

```

Sep 18 20:32
student@nix-commands:~ ping -c 1 -w 1 google.com
PING google.com (142.250.80.110) 56(84) bytes of data.
64 bytes from lga34s36-in-f14.1e100.net (142.250.80.110): icmp_seq=1 ttl=112
time=18.6 ms

--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 18.668/18.668/18.668/0.000 ms

```

Linux Tutorial  
nix-commands.pdf

Assume a simple example: every morning the first thing you do is verify that your most important servers are accessible. You might perform the following commands (but not now):

```

ping mail
ping payroll
ping printer
ping router

```

It would be nice if one command could be entered that would perform all your pings at once. But before we do something like that, let's alter the ping commands to be somewhat friendly for a script. Enter the following:

```

ping -c 1 -w 1
google.com

```

Instead of pinging continuously until interrupted by the user, the above command will ping only once (-c 1), and will only wait one second (-w 1) for the reply. That is an improvement, but you don't really want to see all that output. So try the following (all on one command line):

```

ping -c 1 -w 1
google.com >/dev/null

```

"Up" is displayed on the screen if the ping is successful.

Now you can write a script. Start up an editor, such as leafpad, and enter the following lines:

```

echo
echo "Trying Google"
ping -c 1 -w 1
google.com >/dev/null &&
echo Up

echo
echo "Trying Bing"
ping -c 1 -w 1 bing.com
>/dev/null && echo Up

echo
echo "Trying NPS"
ping -c 1 -w 1 nps.edu >/dev/null && echo Up

```

Save the file in your home directory with the name of "pinger", and then exit the editor.

Make the file executable and try it out by doing the following:

```

11 chmod u+x pinger
11 ./pinger

```

- The ‘ping’ command gives us the ability to check our connection to the outside world. The image shows a ping of 1 ICMP packet to google.com, which results in 0% packet loss, concluding a successful connection test.

The screenshot shows a Linux desktop environment. On the left, a terminal window titled "student@nix-commands:" displays the output of a command to list files in the current directory, followed by a ping command to google.com with the -w 1 option. On the right, a PDF viewer titled "Linux Tutorial" is open, displaying a page about the ping command. The page includes code examples for pinging multiple sites and creating a script to do so.

```

Sep 18 20:41
student@nix-commands:~ ll
total 56
drwxr-xr-x 1 student student 4096 Sep 19 03:37 ./
drwxr-xr-x 1 root root 4096 Jul 7 2019 ../
-rw----- 1 student student 825 Sep 19 03:37 .bash_history
-rw-r--r-- 1 student student 220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 student student 3921 Sep 8 00:37 .bashrc*
drwxr-xr-x 4 student student 4096 Sep 19 03:37 .config/
drwxrwxr-x 1 student student 4096 Sep 8 00:37 .local/
-rw-r--r-- 1 student student 1152 Sep 8 00:37 .profile
-rw-r--r-- 1 root root 0 Sep 8 00:37 sudo_as_admin_successful
-rw-r--r-- 1 student student 220 Sep 19 03:37 Pinger
-rw-r--r-- 1 student student 5410 Sep 19 02:13 listing
student@nix-commands:~$ chmod u+x Pinger
student@nix-commands:~$ ll Pinger
-rwxr--r-- 1 student student 220 Sep 19 03:37 Pinger*
student@nix-commands:~$ ./Pinger
Trying Google
Up

```

Linux Tutorial  
nix-commands.pdf

It would be nice if one command could be entered that would perform all your pings at once. But before we do something like that, let's alter the ping commands to be somewhat friendly for a script. Enter the following:

```

ping -c 1 -w 1
google.com

```

Instead of pinging continuously until interrupted by the user, the above command will ping only once (-c 1), and will only wait one second (-w 1) for the reply. That is an improvement, but you don't really want to see all that output. So try the following (all on one command line):

```

ping -c 1 -w 1
google.com >/dev/null

```

You redirected all the output into a black hole from which nothing returns. So we have removed all the output, but now we don't know if the ping was successful. So, try the following addition (all on one command line):

```

ping -c 1 -w 1
google.com >/dev/null &&
echo Up

```

Save the file in your home directory with the name of "pinger", and then exit the editor.

Make the file executable and try it out by doing the following:

```

11 chmod u+x pinger
11 ./pinger

```

Notice how you had to enter a "./" before pinger? Why is that? The shell is configured to look in given locations for the commands entered by the user. Rarely is the command in the current directory, so if you want to execute something outside the usual places, you must be explicit. The "./" tells the shell to look in the current directory. If you only enter

- Bringing back ‘echo’, we can utilize it when using Leafpad, a text editor that can help create a script and ‘ping’ many sites at once. Once leafpad is opened when then create pings just as before while using ‘echo’ to display text on the terminal. The file is saved to the home directory, where the User permission is granted the power to execute the script. To run the script, ‘./’ is used to tell the shell the script can be found in the current working directory. The outputs spat out are the process in which echo was used to confirm its functionality by saying it's “trying” and then confirming its success by replying “Up”.

## Lab 2: Users

shared login:  
Login timed out after 60 seconds.  
shared login: sudo su  
Password:  
[admin@shared ~]\$ sudo su  
[sudo] password for admin:  
[root@shared admin]# useradd -m bob  
[root@shared admin]# man passwd  
No manual entry for password  
[root@shared admin]# man passwd  
[root@shared admin]# sudo passwd bob  
Changing password for user bob.  
New password:

Sep 21 05:24 users.pdf 85% ▾

Overview 1  
Background 1  
Lab Environment... 1  
Environment 1  
Tasks 4  
4.1 Add user Bob  
Add user B... 2 Use this command to add a user with id bob:  
Add user ... 2 useradd -m bob  
Re-login as... 3 The -m option causes a home directory to be created for the user at /home/bob. Go to the other login terminal and try to log in as the new user bob. Oh, that's right, the user does not yet have a password.  
Add a privi... 3 Passwords are assigned and changed using the passwd program. Use:  
Submission 3 man passwd  
to learn about options for the passwd program. For this lab, we'd like to assign bob a password so that bob can log in, but we want the password to be expired so that bob is forced to change it. Use the passwd program to assign bob a password, and then run the passwd program to cause bob's password to be expired. Then log in as bob and confirm that bob is required to change the password. Do that and make note of the password for later. (For this lab it is OK to write down passwords!)  
Then use exit to log bob out of the terminal.

4.2 Add user Mary  
Add a user with id mary just like you added bob. Before you try to log in as Mary, take a look at a file that Mary will need to access. Use this command:  
ls -l /shared\_stuff/tarts.txt

- After logging into my admin account, I elevate my access to root user using ‘sudo su’ command. Therefore, I can create new users using ‘useradd’ and ‘-m’ adds a home directory for the user in this case, Bob. In addition, using the passwd program allowed me to create a password for Bob to log in. I also used ‘sudo passwd -e Bob’ to expire his password which will prompt him to create a new password when attempting to log in.

shared login: clear  
Password:  
Login incorrect  
shared login: bob  
Password:  
Last failed login: Sat Sep 20 05:47:13 UTC 2025 on pts/1  
There was 1 failed login attempt since the last successful login.  
[bob@shared ~]\$ exit  
logout  
shared login: bob  
Password:  
You are required to change your password immediately (root enforced)  
Changing password for bob.  
(current) UNIX password:  
New password:  
BAD PASSWORD: The password is too similar to the old one  
New password:  
Retype new password:  
Last failed login: Sat Sep 20 05:47:13 UTC 2025 on pts/1  
There was 1 failed login attempt since the last successful login.  
[bob@shared ~]\$ exit  
logout  
shared login: [REDACTED]

Sep 21 05:37 users.pdf 85% ▾

Labtainers 2

4 Tasks

4.1 Add user Bob  
Use this command to add a user with id bob:  
useradd -m bob  
The -m option causes a home directory to be created for the user at /home/bob. Go to the other login terminal and try to log in as the new user bob. Oh, that's right, the user does not yet have a password.  
Passwords are assigned and changed using the passwd program. Use:  
man passwd  
to learn about options for the passwd program. For this lab, we'd like to assign bob a password so that bob can log in, but we want the password to be expired so that bob is forced to change it. Use the passwd program to assign bob a password, and then run the passwd program to cause bob's password to be expired. Then log in as bob and confirm that bob is required to change the password. Do that and make note of the password for later. (For this lab it is OK to write down passwords!)  
Then use exit to log bob out of the terminal.

4.2 Add user Mary  
Add a user with id mary just like you added bob. Before you try to log in as Mary, take a look at a file that Mary will need to access. Use this command:  
ls -l /shared\_stuff/tarts.txt  
Note who owns the file, and the permitted access modes for the owner, the group and other.  
-rw-rw---- 1 frank bakers 8 Apr 29 23:48 /shared\_stuff/tarts.txt  
A brief tutorial on Unix file permissions can be viewed at: <https://mason.gmu.edu/~montecin/UNIXpermis.htm>. For this file, the owner has read and write permission, as do members of the group. Users other than the owner or members of the group have no access to the file. The owner is frank and the

- As Bob I tried logging in before the expire option was put in place and was able to successfully log in. To leave the user account, a simple ‘exit’ is given to the terminal, which logs me out. Once the ‘-e’ or expire option was put in place, and I attempted to log in again I was swiftly prompted to change my password. There were some issues related to password security, in which the Unix password regulation does not allow similar or too short a password, so creating strong password unrelated to the User is very much obliged.

```

shared login: mary
Password:
[mary@shared ~]$ id
uid=1003(mary) gid=1004(mary) groups=1004(mary),1001(bakers)
[mary@shared ~]$ cat /shared_stuff/tarts.txt
2 cups Oreo cookie crumbs
3/4 cup softened butter, divided
1 package (10 ounces) mint chocolate chips
1/2 cup sugar
2 teaspoons vanilla extract
1 cup heavy whipping cream
3 large eggs at room temperature, lightly beaten
Chopped dark chocolate, chocolate curls, whipped cream, optional
[mary@shared ~]$ eggcheck tarts.txt
The tarts.txt recipe has eggs.
[mary@shared ~]$ id
uid=1003(mary) gid=1004(mary) groups=1004(mary),1001(bakers)
[mary@shared ~]$ touch newfile.txt
[mary@shared ~]$ ls -l
total 0
-rw-rw-r-- 1 mary mary 0 Sep 21 12:44 newfile.txt
[mary@shared ~]$ newgrp bakers

```

- In the same manner Bob was created Mary was also added as a user using ‘useradd’. Mary had to pertain to a certain group to access certain files to perform her baker duties. On the admin terminal, ‘usermod’ was used to add Mary to the baker group. So on the Mary terminal, we can input ‘id’ and it now displays bakers as part of one of Mary’s groups that she is in. Resulting in file access and being able to view the shared info.

```

There was 1 failed login attempt since the last successful login.
[bob@shared ~]$ cat /shared_stuff/tarts.txt
cat: /shared_stuff/tarts.txt: Permission denied
[bob@shared ~]$ eggcheck tarts.txt
Traceback (most recent call last):
  File "/usr/bin/eggcheck", line 11, in <module>
    with open(fname) as fh:
IOError: [Errno 13] Permission denied: '/shared_stuff/tarts.txt'
[bob@shared ~]$ cat /tmp/tmpfile.txt
2 cups Oreo cookie crumbs

3/4 cup softened butter, divided

1 package (10 ounces) mint chocolate chips

1/2 cup sugar

2 teaspoons vanilla extract

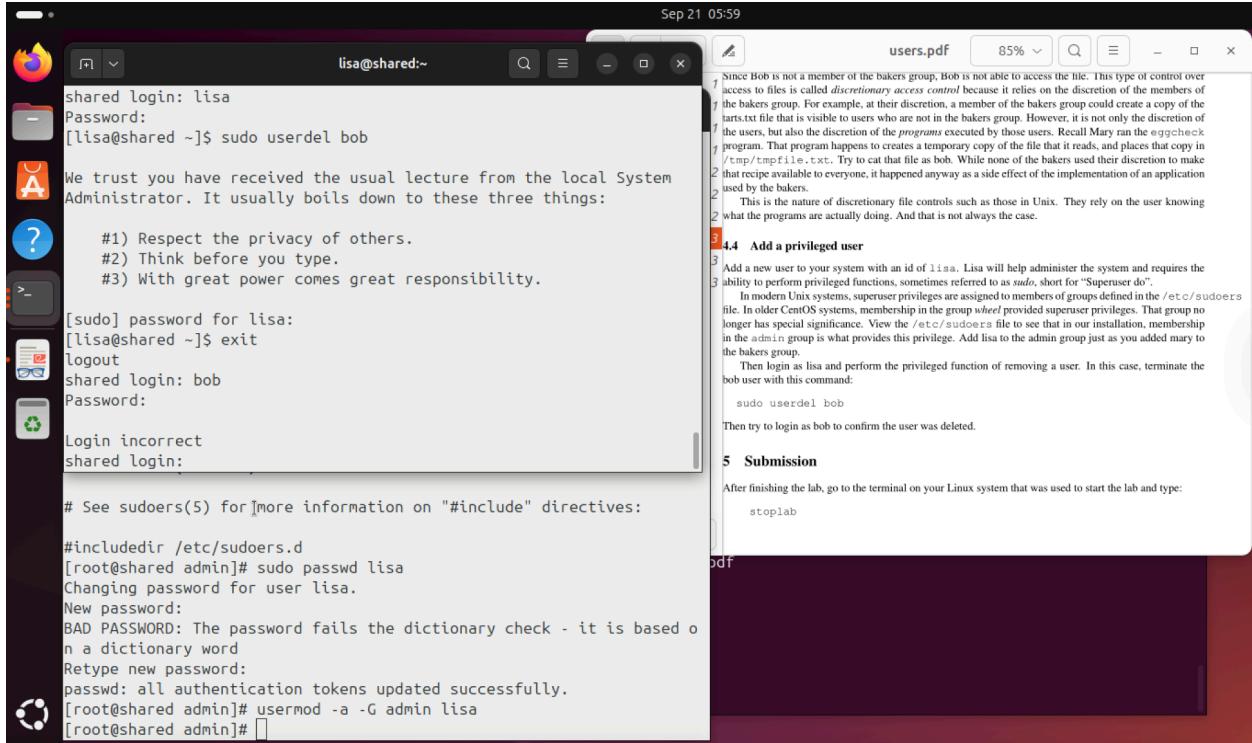
1 cup heavy whipping cream

3 large eggs at room temperature, lightly beaten

Chopped dark chocolate, chocolate curls, whipped cream, optional

```

- Here Bob is not part of any group and can not access the same files as Mary due to the discretionary access control that the bakers group have set up. But when Bob runs the same egg check program as Mary, he is denied permission due to his status, but the program has already ran and stores a temporary file on his system, but it was not shown on the terminal due to his status. But due to the program creating a temporary file, Bob can access that file with a simple ‘cat’ command, a big downside to bad configuration of a program in regards to its original set of permission rules.



- Lastly, Lisa was a user created with privileged access to help with administrative work. She was added to the special file of sudo users '/etc/sudoers/' with the command 'usermod -a -G admin lisa'. When logging into Lisa, she can now use the 'sudo' command to escalate her power and use certain commands. In this example, she can do 'sudo userdel bob', which removes Bob from the system completely.

```
[root@shared admin]# sudo passwd -S bob
passwd: Unknown user name 'bob'.
[root@shared admin]#
```

- To confirm, we can check the status of Bob's password, which returns unknown user, meaning that there is no longer a user under Bob. You can also try logging into Bob's account, but it will not be successful.

## Reflection Section

1. How do Linux file permissions contribute to system security?

- File permissions play a major role when it comes to system security because one of the major concepts of cybersecurity is confidentiality, meaning files should always be set up so that only those who should know, do know. Unmanaged permissions can lead to open vulnerabilities that threat actors can exploit and cause damage. With file permissions, users, groups, and other groups are restricted on what they can read, write, or execute.

2. What are the potential security risks of improper user management?

- Improper user management can lead to a whole list of negatives, such as unauthorized access for non-users to sensitive data, or you can have users who weren't configured correctly being able to escalate their own privileges. Strong user management will prevent these events from happening, and it secures users from small things like password management to user permissions.

3. How might an attacker exploit weak file permissions during a penetration test?

- An attacker might exploit weak file permissions in many ways. For example, bad file configuration can have unwanted users or groups accessing confidential files, and if the wrong permissions are set, these files can be overwritten or data can be stolen. An attacker can simply go around opening files using the 'cat' command based on the file permissions displayed when 'll -a' is used.

4. Why is command-line proficiency essential for cybersecurity professionals?

- The CLI is home to lots of cybersecurity tools, such as nmap, tcpdump, Wireshark, etc.

As well as its versatility with a lot of different OS models, it is very common and being able to understand the CLI will give an edge against, say a GUI in terms of efficiency and precision.

5. What challenges did you encounter during the labs, and how did you overcome them?

- Some challenges I faced when completing the labs were understanding the switches that follow commands. I was able to figure my way through by using the ‘man’ command to open up the manual and read up on the command and its functionality. There were also times of stumbling over the same issue, but taking a quick break or some time to refocus and back track to understand where I went wrong has helped instead of simply trying to push through with no progress.

6. How do these fundamental skills relate to advanced ethical hacking techniques?

- In advanced hacking, many techniques are built upon the foundations of the command line, file permissions, scripting, and privileged escalation. Understanding these everyday concepts allows for critical thinking to then perform better techniques that can promote more efficient and precise results.

### Concluding Summary

The Labtainer exercises helped me develop core competencies in user administration, file system management, scripting and command-line navigation. Thus helping me prepare for more advanced ethical hacking techniques that require understanding how these topics function and work alone or together. Creating users such as bob, mary and lisa has exposed me to file permissions and management in coordinating users into groups or specific roles. This provides insight on how defense against attackers can be set up but also how vulnerabilities made present themselves if not careful. When playing with ping it helped me learn how to quickly diagnose connection issues or even create my own script that can automate the process of connectivity to a number of servers. Moving forward, I will continue to build on these fundamentals through scripting, regular permission audits on my personal device, and controlled lab practice so I can safely progress to more advanced techniques to ensure the confidentiality, integrity, and availability of data for clients and further develop my career in cybersecurity.