Linux Primer

The labs in this series are executed using Linux command shells. Therefore, it is important to know how to open and use a command shell in Linux.

Run Linux

In the labs in this series Linux machines will exist as Virtual Machines (VMs) in VirtualBox. To continue in this primer it is recommended that you complete the part of Lab 1 where VirtualBox is installed and set up with two VMs and then return to this primer. You will then be able to work in an environment that is suitable for this primer. Start a VM as stated in Lab 1: open VirtualBox on a host system (VirtualBox is supported in Linux, Windows, and MacOS) to get the VirtualBox main page as shown in Figure 1 where Ubuntu and Kali systems are available, highlight the Ubuntu VM as shown and click the Start arrow.

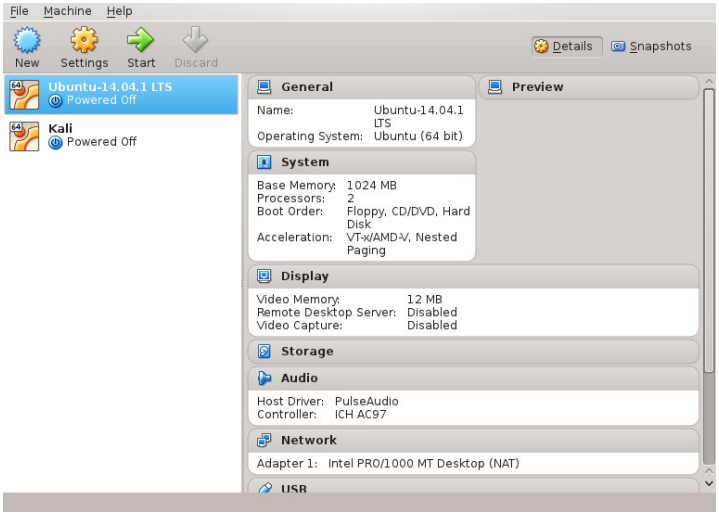

Figure 1: VirtualBox main page with two VMs loaded

When you have booted into Ubuntu click ‘Activities’ in the upper left area of the VM and type ‘terminal’ in the text field as shown in Figure 2, then click the terminal icon (shown in Figure 2 where the mouse cursor is). The resulting terminal (alternatively called the console) is shown in the center of the desktop in Figure 3. Click over the terminal to activate it so it is ready for commands. All commands are run by first typing the command on a blank terminal line to the right of the prompt and then typing the ‘Enter’ or ‘Return’ key. The prompt shows up on the left side of the terminal window and typically ends in `$`. The prompt in Figure 3 is `franco@franco:~$`.

ls:

The first command to experiment with is the one that shows the names of files in a directory. With the terminal window active, type `ls` then hit the ‘Enter’ or ‘Return’ key. The `ls` command, like all commands, takes options and arguments. An option allows the command to produce a different kind of output. An example of running `ls` with an argument (and no options) is ‘`ls /sbin`’. This command lists the names of files in the directory `/sbin` as shown in Figure 4. Those files are commands that may be run in the terminal. Other directories that contain commands are `/usr/sbin`, `/bin`, and `/usr/bin`. You can get a long listing with ‘`ls -l /sbin`’, an example of which follows the figures below.

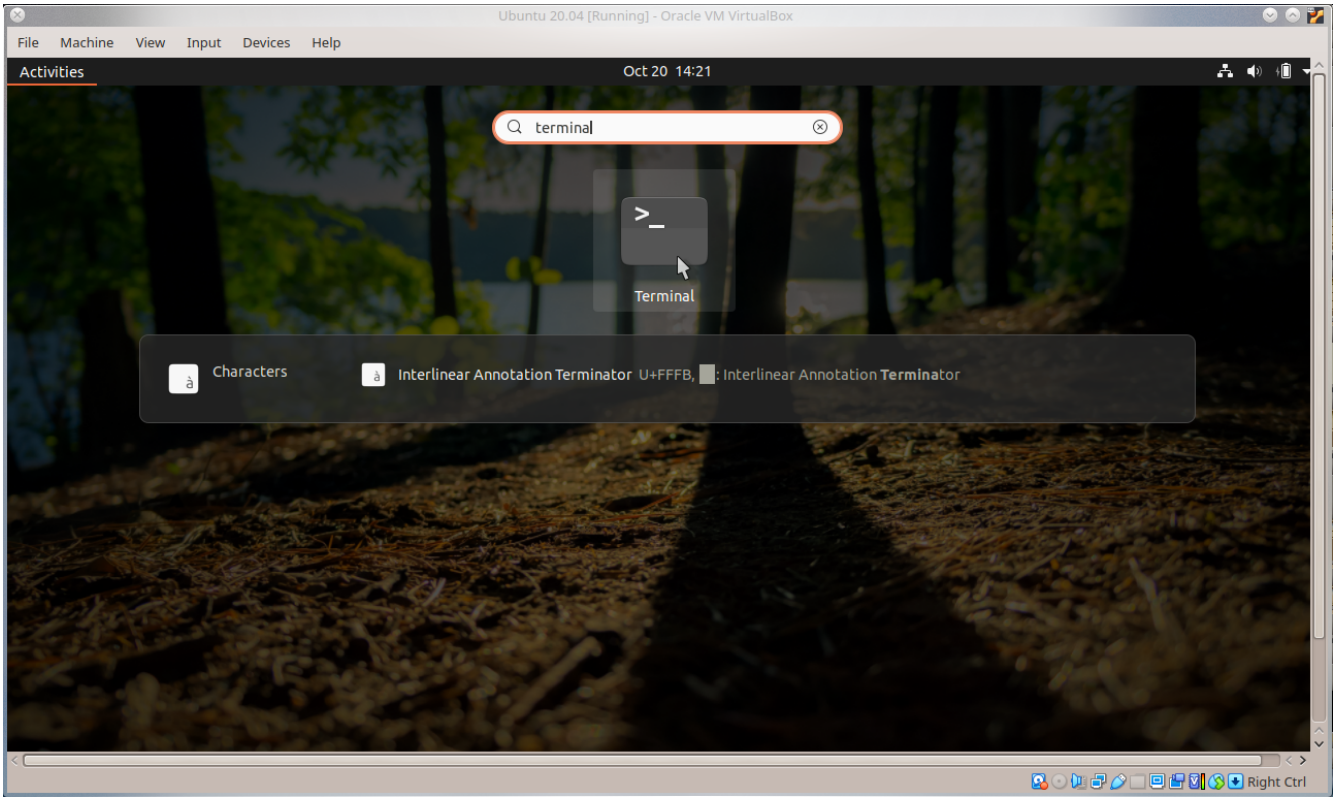

Figure 2: Ubuntu VM desktop after 'Activities' is clicked.

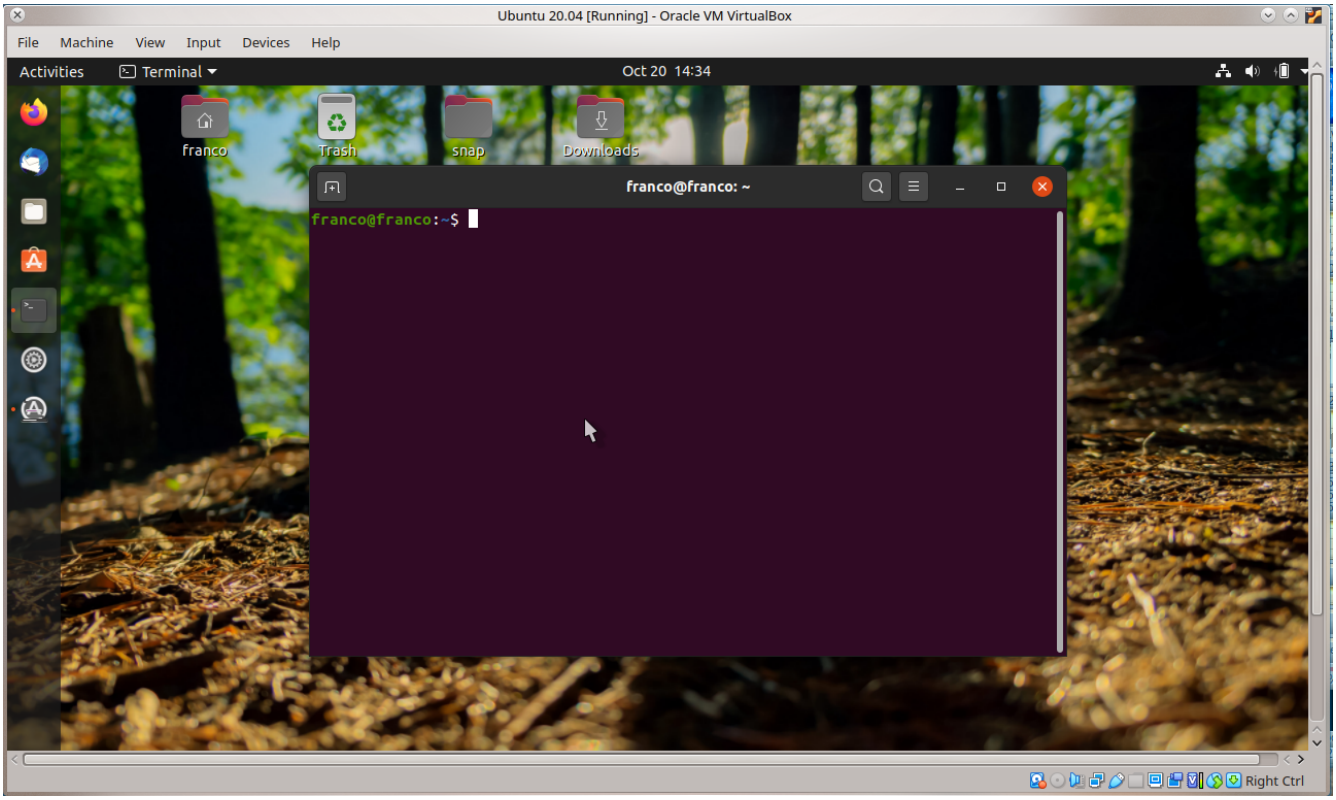

Figure 3: A terminal has been opened in the center of the desktop. Click over the terminal to make it active and ready for commands.

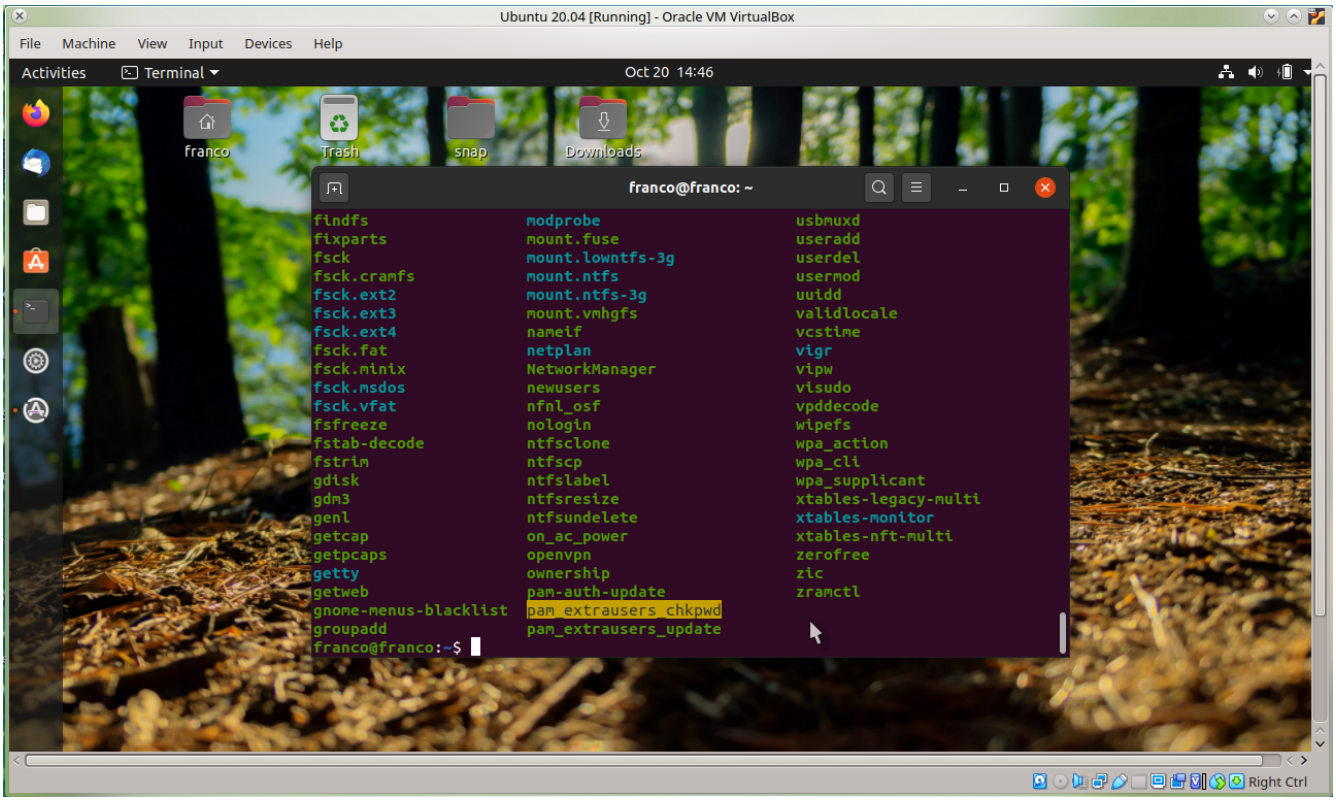

Figure 4: Result of running the command '`ls /sbin`' from the terminal

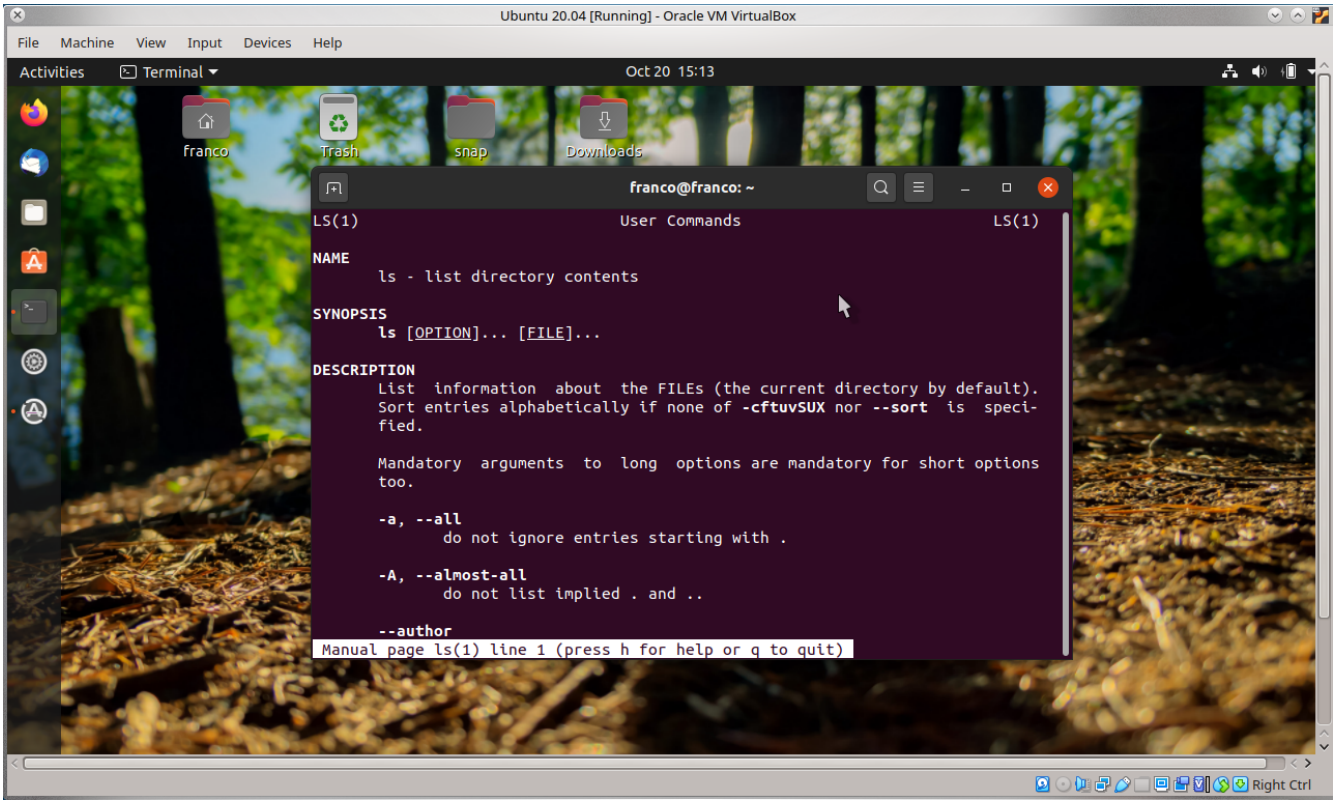

Figure 5: The result of running '`man ls`'

There is one line for each file in a long listing. You probably recognize some of the fields that are on such lines, for example last date and time of modification and the size of the file. But you may not know what the leftmost four fields are and these will be explained now with the following long listing example from directory `/usr/sbin`:

```
-rwxr-xr-x 1 root root  220488  Feb 28   2020  xtables-nft-multi
```

In Linux there is the user, the group, and everyone else. According to the long listing above, the `xtables-nft-multi` file belongs to user `root` and is in group `root` as well. Depending on permissions that are set for the file users may or may not have permission to read, write, or run the file as a command. The permissions are listed in the leftmost field which consists of 10 characters. The rightmost set of 3 characters identifies the permissions for everyone that is not the stated user (`root` in this case) and is not in the stated group (`root` in this case). If the rightmost 3 characters are `rwx` in that order then anyone can read, edit, and run the file as a command. If the 3 characters are `r-x` then anyone can read and run the file but anyone other than the stated user who is not in the stated group cannot edit it. If the 3 characters are `r--` then anyone other than the stated user who is not in the stated group can read the file but cannot edit or run it as a command. If the 3 characters are `---` then anyone other than the stated user that is not in the stated group cannot do anything with the file. Unless you are `root` you cannot edit `xtables-nft-multi.`

The leftmost character of the leftmost field indicates a property of the file. For example, if instead of `-` the leftmost character is `d` then the file is not a file but a directory. To find out who you are run the command `whoami`. To find out the current date and time taken from your system clock run the command `date`.

Please do not run any commands without knowing what they do first. To find out what a command does, use the `man` command.

man:

To find out about the `ls` command run '`man ls`' to get what is called a *man page* for command `ls`. This is shown in Figure 5. A man page is typically very long and contains several sections occupying several screens. The screen showing in Figure 5 contains Sections 'Name', 'Synopsis', and part of 'Description'. Without going any further you can see what the `ls` command does. You can see the rest of the man page by scrolling one line at a time with the down arrow key or a screen at a time with the 'pg dn' key. You can search for a string of text by typing slash (`/`) followed by the string. To repeat the search type slash followed by Return. Options are identified by a minus (`-`) or double minus (`--`) that appears before them. For example, for `ls` the `-C` option is invoked to list entries by column. You cannot run commands from a man page. To leave a man page type '`Q`' or '`q`' (there is no need to follow this with a 'return').

Exercise 1:

Search the `ls` man page for the option that outputs object names in every directory beneath the current directory. What is that option?

Exercise 2:

Search the `ls` man page for the option that orders object names by size. What is that option?

Exercise 3:
Use that option to list files in directory `/etc` in long listing format. What was the result? Write a few of the lines that you saw in the order you saw them – top to bottom.

Other Very Useful and Basic Commands

pwd :

This command shows the *current working directory*. The reason for directories is: 1) the tens of thousands of files on a system need to be organized so a user can easily see a listing of files that are important at the moment without clutter from the other files on the system and 2) searching for a file becomes easy and fast. At any moment a user has direct access to objects in the *current working directory*. Running `pwd` shows what that directory is. For example, running `pwd` might return `/home/franco`. This is decoded as follows: the directory `/` is known as the root directory. All directories and files can be reached from `/` as will be shown below. Run command '`ls /`' and get the result shown in Figure 6. Some of the items shown in that figure are directories (in darker blue), some are symlinks (in lighter blue and discussed below), and one (in white) is a file. Notice the item `home` which is a directory. That directory potentially contains more directories and files. To see what is in that directory run '`ls /home`'. For me, this yields only the object named `franco` which is a directory. So, directory `franco` is 'under' directory `home` which is under directory `/`. This so-called 'path' to directory `franco`, my home directory, is denoted `/home/franco`.

Exercise 4:
What is the full path to your current working directory?

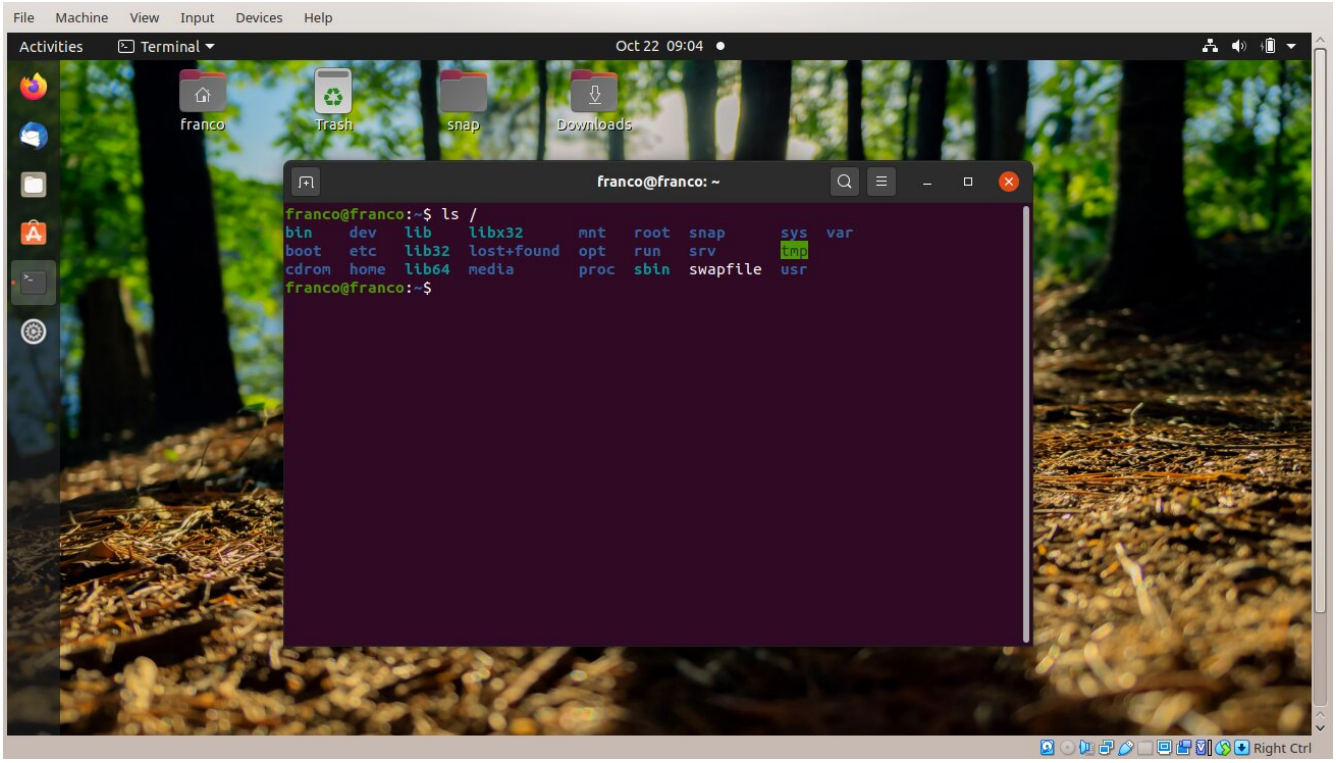

Figure 6: Listing of the directories and files in the root directory

`file:`

To determine the type of an object. There are numerous types of objects and colors are not a good way to determine the type of an object so the `file` command is used for that purpose. Here is an example – run this:

```
file /usr
```

The result is

```
/usr: directory
```

Exercise 5:
Find the type of file for the following objects:

```
/usr/share/pixmaps/faces/yellow-rose.jpg
/usr/share/mime/application/vnd.adobe.flash.movie.xml
/swapfile
/var/lib/colord/mapping.db
/bin
/usr/bin
/usr/share/doc/util-linux/cal.txt
/usr/share/doc/util-linux/howto-pull-request.txt.gz
```

`mkdir:`

To create a new directory use `mkdir`, for example as follows:

```
mkdir src
```

If running `ls` before the above command produces

```
Downloads  snap
```

then after running the above `mkdir` command `ls` will return

```
Downloads  snap  src
```

`mkdir` will succeed only if you have permission to make the directory you wish to make. You can use '`ls -l`' to determine that. For example, suppose your username is `franco` and you wish to make directory `src` in directory `/usr`. Then run '`ls -al /usr`' and get something that starts out like this:

```
drwxr-xr-x 14  root root   4096  Apr 23  03:34 .
drwxr-xr-x 20  root root   4096  Sep  1  05:46 ..
drwxr-xr-x  2  root root  40960  Oct 22  09:52 bin
drwxr-xr-x  2  root root   4096  Apr 23  03:35 games
...
```

It is the line associated with '`.`' on the right that shows the relevant permissions. The '`.`' means the directory that was asked for, which in this case is `/usr`. For user `franco` to be able to create directory `src` in `/usr` the permission that is needed is write permission which is identified with `w` in the appropriate three letter section on the far left. Since `franco` is not `root` and is not in group `root` the rightmost three letters of the permissions string applies. This is '`r-x`'. The absence of '`w`' means `franco` cannot create directory `src` in `/usr`.

Exercise 6:
Determine that a directory can be created in your home directory. The home directory does not have to be the current working directory to do this. Then create a directory called

`test_directory` in your home directory. Then create a directory called `first-level` in `test_directory`.

rmdir:

Deletes a named directory. This only works if the directory is empty (contains no files and no directories). If a directory is to be deleted with files and directories contained within it command `rm` is used with the `-rf` option (see below). Example: run these commands:

```
mkdir tester
ls
rmdir tester
ls
```

The directory `tester` is seen to exist with the first `ls` but not with the second `ls`.

cd:

Change the current working directory. Run `cd` with no arguments to change the working directory to your home directory. Run `cd` with a path to a directory to change the current working directory to the named path. For example,

```
cd /usr/share/pixmaps
```

changes the current working directory to `/usr/share/pixmaps`. If the target directory does not have the '`x`' permission bit set, you will not be able to change to that directory. If the '`r`' permission bit is not set you will not be able to read the directory. If the relevant permissions for the target directory are '`r-x`' you will be able to read the directory but not write to it (the directory is read only).

Exercise 7:

Make the current working directory `/etc`. Determine whether you can create a directory in `/etc`. Make the current working directory `/etc/init.d`. List the files in the current working directory.

touch:

Change the modified time of an existing file or create a new file. Change the current working directory to your home directory. Then run:

```
touch new-file.txt
```

Then run this:

```
ls -l new-file.txt
```

You should see something like this:

```
-rw-rw-r-- 1 franco franco 0 Oct 22 13:21 new-file.txt
```

This shows new-file.txt now exists but has 0 bytes of content.

Exercise 8:

Change directory to `test_directory/first-level`. If that directory does not exist create it. In that directory create a new file called `new-file.txt` with `touch`. Use `ls` to find out how many bytes `new-file.txt` contains.

echo:

Create or add text to a file. For example, in your home directory do this:

```
echo "forty barrels of oil" > new-file.txt
```

causes `new-file.txt` to contain 21 bytes of text as can be seen with `ls -l new-file.txt`. The operator > redirects the text output by `echo` to the file. Doing this:

```
echo “fifty barrels of oil” > new-file.txt
```

replaces everything that was contained in `new-file.txt` with ‘`fifty barrels of oil`’. The >> operator redirects and appends. Therefore,

```
echo “on the wall” >> new-file.txt
```

causes new-file.txt to contain the following:

fifty barrels of oil
on the wall

Exercise 9:
Add the following content to `test_directory/first-level/new-file.txt`:
We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil

Then add “*Just Kidding*” to that file.

cat:

Concatenate files and print to standard output. To see what the contents of `new-file.txt` is from the directory in which it exists run this command:

```
cat new-file.txt
```

Exercise 10:
Display the contents of `test_directory/first-level/new-file.txt`

cp:

Copy a file or contents of a directory to another location. Change directory to your home directory. Create directory `first`. Create directory `first/second`. Then run these:

```
cp new-file.txt first/second
cat first/second/new-file.txt
```

The `cp` command can also copy the contents of a directory to another location with the option `-r` like this:

```
cp -r first/second .
```

which copies all of directory second to the home directory (the ‘`.`’ means the target is the current working directory which, in this case, is the home directory).

Exercise 11:
Copy all of directory `test_directory` to a new directory named `first`. Then display the contents of `new-file.txt` which is somewhere in the hierarchy of directory `first`.

rm:

Removes files and or directories. Run the following commands from the home directory:

```
rm new-file.txt
ls
```

You will notice that file `new-file.txt` no longer exists in the home directory. Now run the following commands:

```
rm -rf first
ls
```

Observe the directory `first`, which contained directory `second` and file `new-file.txt`, no longer exists in the home directory. The `-rf` option (which is actually two options squished together) means force (the `f`) everything in `first`, including `first` (the `r`).

mv:

Like `cp` except that the original files and or directories are erased as though `cp` then `rm -rf` are applied. So, the following command:

```
mv second first
```

renames `second` to `first`.

Exercise 12:

From the home directory, rename directory `first/first-level` to `first/second-level`. Then display the contents of `first/second-level/new-file.txt`.

sudo:

Acquire the privileges of the system administrator or run a command with the privileges of the system administrator. This command works for a user only if there is an entry in the file `/etc/group` like this: `sudo:x:27:username`. To acquire system administrator privileges do this:

```
sudo su
```

to run a command with system administrator privileges do this:

```
sudo command
```

The user will be asked for its password, then the command will complete. One of the most important uses of `sudo` is updating the operating system and applications. This is done in two steps with Ubuntu.

Step 1:

```
sudo apt update
```

Step 2:

```
sudo apt upgrade
```

The `sudo` command also allows software installation, for example like this:

```
sudo apt install net-tools
```

The above will install network tools including ifconfig which is needed for the next exercise.

Exercise 13:

Find a network interface over which traffic flows. This can be done with `/sbin/ifconfig`. Consult the man page of `ifconfig` for help in using it. Use the man page for `tcpdump` to find out how to list packets that pass through that interface. Now apply `tcpdump` to that interface. What happened? Now correct the problem with `sudo`.

chown:

Change group and ownership of files and directories. Most often used with `sudo`. For example, suppose the result of `ls -l` includes this:

```
drwxr-xr-r  franco franco  4096 Oct 22 08:16 snap
```

then, after running `sudo chown root.root snap`, `ls -l` includes this:

```
drwxr-xr-r  root root 4096 Oct 22 08:16 snap
```

So, ownership has changed from franco to root and the group has changed from franco to root as well.

chmod:

Changes permissions of a file or directory. Users may change permissions of files or directories they own. Only one way to operate this command is given here but the man page of `chmod` can be consulted for other possibilities which are more advanced. Recall a permissions string has 10 characters with three groups of three characters on the right which express read, write, and execute permissions for users who are owners, groups, and all else. In the `chmod` command the letter 'u' corresponds to owners, 'g' corresponds to group, and 'o' corresponds to all else. The following shows this correspondence on a typical permissions string where rwx strings account for permissions of each of owner, group, and all else in that order from left to right and the leading underscore is just a placeholder for the leftmost character that is not being considered here. See the man page of `chmod` for information about that leftmost character:

```
_rwxrwxrwx
 \.../ \.../ \.../
  u   g   o
```

The `_` on the left is a place holder and stands for `-` or `d` or some other characteristic. Running

```
chmod o-rw <some-file-or-directory>
```

where the current permissions string is as above results in a permissions string that looks like this

```
_rwxrwx--x
```

that is, read and write permissions are removed from users other than owner and group. The letter 'a' corresponds to all three classifications. Run

```
chmod a-rw <some-file-or-directory>
```

where the current permissions string is the first one above to get this:

```
_--x--x--x
```

Permissions can be added as well as subtracted. Run this

```
chmod ug+rw <some-file-or-directory>
```

on the file or directory with read and write permissions denied for all categories to get this

```
_rwxrw-x--x
```

Exercise 14:

File `/usr/bin/dumpcap` must be a set user id program in order to be used by `wireshark`. But the default permissions for that file are `rwxr-xr-x`. Consult the man page for `chmod` to figure out how to change permissions to `rwsr-xr-x` with `chmod` and attempt to make the change without using `sudo`. What happened? Now use `sudo` then

```
ls -l /usr/bin/dumcap
```

to make sure the permissions are as needed. If you feel lucky, attempt to use `wireshark`.

Command Line Editing

The command line can be edited. This is done when either the command being typed has an error that needs to be corrected or when reusing a previous command that needs to be

changed slightly. A history of commands is maintained in a buffer and can be viewed with the command `history`. The commands in the history buffer are given index numbers so it is easy to refer to them. A user may navigate to a position in the history buffer to repeat or edit a command that had been run earlier – the lower the index number, the earlier the command was run. There is also a command cursor that may occupy any position in the currently displayed command. Here is a list of line editing commands:

Command	Description
⬆ (up arrow)	Display the command of one lower index number (if possible)
⬇ (down arrow)	Display the command of one higher index number (if possible)
➡ (right arrow)	Move the command cursor one character to the right (if possible)
⬅ (left arrow)	Move the command cursor one character to the left (if possible)
^A (control A)	Move the command cursor to the beginning of the command
^E (control E)	Move the command cursor to the end of the command
^K (control K)	Cut all command characters from the cursor to the end of the command
^Y (control Y)	Restore the most recently cut characters beginning at the command cursor
^D (control D)	Delete the displayed command character under the cursor, cursor stays put
!<index-number>	Display the command with history index <index-number>

Character Strings With Meanings

Character(s)	Meaning
`.`	Current directory – to run a command, say a script, that is in the currently directory use `./<command>` (this is necessary for security purposes)>
`..`	Parent directory (e.g. parent directory of `first/second` is `first` - from directory `second`, `cd ..` makes the current working directory `first`)
*	String of any number of any characters (e.g. `ls *.jpg` prints to the console all files with a `jpg` extension).
?	Any character (e.g. `ls ???.jpg` prints to the console all `jpg` files whose name is 3 characters long up to the extension).
“	Surround a string so that spaces are considered part of a single string. See the next section for examples.
‘	Surround a string to include escape characters and spaces as a single string. See the next section for examples.
\	Escape character used to add characters like #, &, and even \ to a single string. See the next section for examples.
~	Home directory (e.g. `ls ~` lists all objects in the home directory)

Searching

Three principal tools are used for finding objects as follows.

grep:

Find a file that contains a string. The following prints the names of all files with `txt` extension in the current working directory that contains the string '`address`' along with the line(s) containing the string `address`.

```
grep -H address *.txt
```

The `-H` option prints the name of the file. The following prints the names of files, plus lines containing '`address`' in the current working directory and all subdirectories of the current working directory:

```
grep -rH address *
```

The `-rH` option causes this deep search and prints the file name. The following prints the names of `.txt` files and lines containing the string `Madame Bovary` in any subdirectory of the current working directory:

```
grep -H "Madame Bovary" */*.txt
```

The following prints all files in subdirectory `first` that contain the string `#$` which must be escaped in this context:

```
grep -H '\#\$' first/*
```

See the man page for `grep` for many more options.

locate:

Locate an object whose name contains the given string. To use `locate` package `mlocate` must be installed – see the `sudo:` section for an example of using `apt` to install a package. This command makes use of a database that can be manually built with the command

```
sudo updatedb
```

With a current database you can use locate like this, for example:

```
locate .txt
```

which will likely produce far more results than you want. To limit the output pipe the result to `grep` as shown in the Piping section below.

find:

Search for files under very many conditions concerning properties. An example of the most basic usage of the find command is the following:

```
find . -name new-file.txt
```

this command finds, from the current working directory ('`.`') all the locations of all files named `new-file.txt` in subdirectories of the current working directory. Doing this:

```
find . -name "*.txt"
```

finds all file with the `.txt` extension. Once an object is found by find it is possible to apply some command to that file. Here is a simple example:

```
find . -name "*.txt" -exec cat {} \;
```

`find` supplies the files, then for each file supplied, `-exec` runs `cat` on that file ('`{}`'). The command in this case must end with `\;`. There are very many options for find – please check the man page for more. Here is a more advanced example:

Exercise 15:

What does this command do?

```
find [A-Z,0-9,a-z]* -maxdepth 3 -type d \( ! -name . \) -exec ls {} \;
```

Exercise 16:
Use `find` to find broken symlinks in subdirectories of the current directory.

Piping

A pipe is denoted by '`|`'. Pipes are used as a convenient way to send the output of one command to the input of another. A simple example for determining the number of objects in a directory, in this case `/usr/bin`, uses a command `wc -l` which counts the number of lines it sees as inputs, is the following (this is not the only or preferred way of doing it – just a simple way to illustrate a pipe):

```
ls -l /usr/bin | wc -l
```

The output of `ls -l /usr/bin` is a list of objects in `/usr/bin`, one line for each object. You have seen something similar to this in section `ls:`. That output becomes the input to `wc -l` which outputs the number of input lines given to it. Check the man page for `wc` for more options for that command. Thus the result of the above command is the number of files in directory `/usr/bin`. A common use of `|` is to send text to command `less`. The `less` command allows a user to view a piece of a text file that fits into a terminal window. A user can apply command `cat` (see section cat:) to print the entire contents of a text file but if the user wishes to see a frame of text at a time the output of `cat` can be piped to `less` like this:

```
cat big-text-file.txt | less
```

This allows the user to scroll through the `txt` file one line or one page at a time. Here is another relatively simple example making use of the command `awk` which is extremely useful in creating scripts but is beyond the scope of discussion here:

```
echo `pwd` | awk '{print substr($1,14)}'
```

The backward quotes around pwd cause `pwd` to be invoked (see section pwd:) which produces a full path to the current directory. The command `awk` is applied to this output with the option to remove the first 14 characters of the path (`print substr($1,14)`). This example is shown so you realize there is a lot of expressive power waiting to be used on the command line. A more advanced primer can be consulted for more interesting examples and, of course, there are always the man pages to check.

Exercise 17:

What does the following do?

```
locate .txt | grep -v /var | grep -v /etc | grep -v /usr
```

Editing

A useful text editor in Linux is `gedit`. This is a what-you-see-is-what-you-get editor. You will not have to use the man pages to figure out how to use it as menus are provided for navigation. To run `gedit` just type `gedit` in a terminal window then hit Return or Enter. Alternatively, click 'Activities' and type 'editor'. Another popular text editor is `nano`. Run `nano` as a command from a terminal window. Help is provides at the bottom of the editor window so it is fairly easy to use. An editor that always comes with Linux is `vi`. This is not recommended for beginners.

VirtualBox – Some notes on the ‘Network’ selection in ‘Settings’

NAT

Selecting NAT for your adapter to be attached to will provide an IP address like 10.0.2.15. Two VMs that are attached to NAT will have the same IP address so NAT is not to be used to create a network of VMs. However, NAT can be used to communicate with nodes on the Internet as well as your host. Thus, use NAT to update your VM and use NAT to send and receive files from your host and anywhere outside your host. Use NAT to browse the Internet.

NAT Network

Set your adapter to ‘NAT Network’ to form a local network of VMs. All these VMs will have different IP addresses and will be able to communicate with each other. All these VMs will be able to communicate with the host and Internet nodes. A browser can reach Internet pages. A NAT Network may have to be created. To do this pull down the ‘File’ menu in VirtualBox and select ‘Preferences’, then select ‘Network’. If no networks are named, click the green box on the right. This creates a line that contains a name (usually NatNetwork) which is checked as enabled. This can be edited by clicking the bottom square on the right (the middle square removes the network). The edit box allows a custom configuration including range of IP addresses, IPv6 support, network name, and port forwarding.

Host Only Adapter

The above two options do not add any network interface to the host. With host only adapter a network is created that includes all the VMs plus the host. A VM attached to a host only network can communicate with the host and other VMs but not the Internet. A new network interface is created on the host to allow connection to the VMs. To create a host only network pull down the ‘File’ menu in VirtualBox and select ‘Host Network Manager’. Click create. In the edit window choose IP address for the host, network mask, if dynamic addressing is use click the enable button and create the DHCP server by clicking the ‘DHCP Server’ tab and choosing the address range. For a VM, click ‘Host Only Adapter’ and choose one from the menu. Even if the VM is no running the name of the network becomes the name of a network interface that is active in the host. Starting a VM with host only adapter results in an IP address that is on the same network as the host.