Learning the Linux Command Line

A beginners guide

First Edition

MARTIN PETRAUSKAS

Northeastern University
Khoury College of Computer and Information Sciences

How To Use This Guide

This guide is for anyone who wants to learn the Linux operating system and the Bash command line. Throughout the guide, there are examples of how to use various bash commands. The chapters cover particular topics ranging from manipulating files and directories to connecting to remote servers.

Each chapter has basic definitions, examples, and a command summary that contains information about the specific topic. Furthermore, in each chapter, commands are printed in **bold**, files, filepaths, and command syntaxes are in `this font`, and examples that could be typed in the command line are in *italics*. The command summary at the end of each section contains a brief overview of the new commands introduced in that chapter.

If there is trouble running the examples given in this guide, it could be one of the following issues. First, a file might not be present. Some commands, such as filtering commands, require a particular file as input. Second, the syntax for the command could be incorrect. Commands are sensitive and some require additional arguments to run properly. Double check to ensure that all the necessary arguments are present and that nothing is mistyped.

Acknowledgments

There are a great number of people who helped collaborate on this guide. I would first like to thank Professor Christo Wilson for his support of the project. I came to Christo in mid-2018 with the idea of writing this guide. He supported the project from the beginning and he assisted with formulating some of the examples that are seen in the guide and making sure all the relevant information is covered.

I would also like to thank Samir Elhelw, Walter Geanacopoulos, Matthew Kline, and Garrett Tucker for reviewing this guide. And lastly, I would like to thank my father, Bruno Petrauskas, for proofreading and editing this guide from the very beginning.

Contents

1 The Virtual Machine **7**

2 Basic Navigation of the Command Line **21**

3 File Basics **29**

4 File Manipulation **35**

5 File Permissions **41**

6 Manual Pages **47**

7 Complex Inputs **51**

8 Process Management **57**

9 Filters and Regular Expressions **65**

10 Input Output Redirection **71**

11 Basic Networking **75**

12 Command Cheat Sheet **79**

Chapter 1

The Virtual Machine

Learning to use the Linux operating system requires the proper software and a Linux distribution. The examples in this guide use the Ubuntu operating system on a virtual machine which is installed on the Oracle VM VirtualBox virtualization software. This section will step through the process of configuring a virtual machine and installing Ubuntu on the machine.

STEP 1: Acquire a Linux Operating System

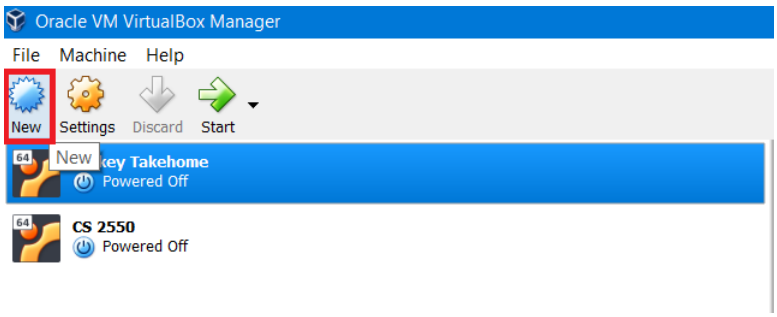
Download the Ubuntu OS. The current version is Ubuntu 18.04 LTS, and it can be downloaded from the official Ubuntu website: https://www.ubuntu.com/download/desktop. The download file is large, approximately 1.8GB, so downloading the file can take a while.

STEP 2: Acquire the Virtual Machine Software

Download the necessary virtualization software to create a virtual machine with Ubuntu. The Oracle VM VirtualBox can be downloaded from the VirtualBox website: https://www.virtualbox.org/wiki/Downloads. Once the download is complete, install the virtualization software on your computer before proceeding to Step 3.

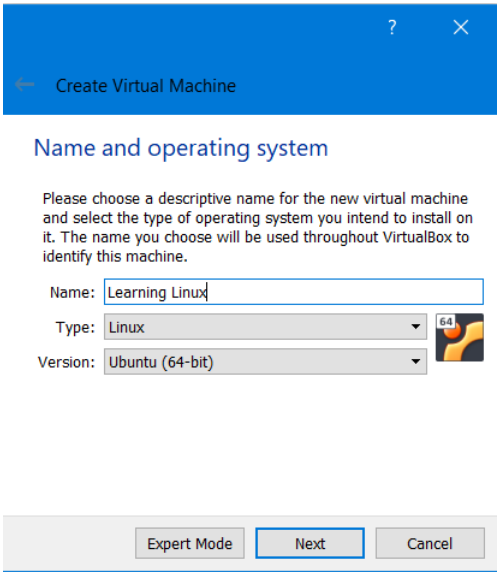
STEP 3: Create a New Virtual Machine

Start the VirtualBox program. In the upper left corner, click on the "New" button.



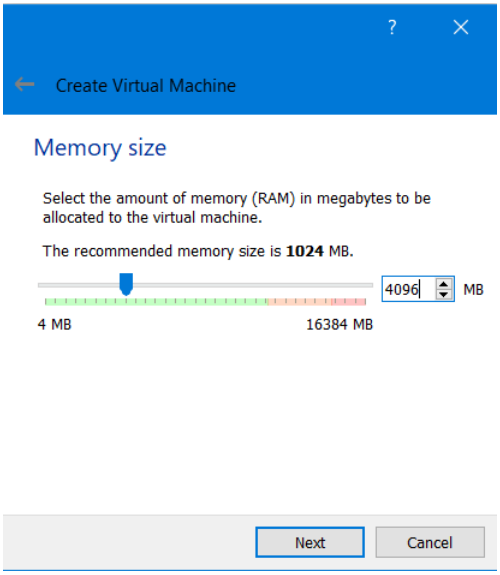
STEP 4: Specify Virtual Machine Name and Type

Enter a simple name that describes the purpose of the virtual machine. Next, select "Linux" as the option for the type of machine. Then, select the proper Ubuntu version (32-bit or 64-bit). Click "Next" after entering the information.



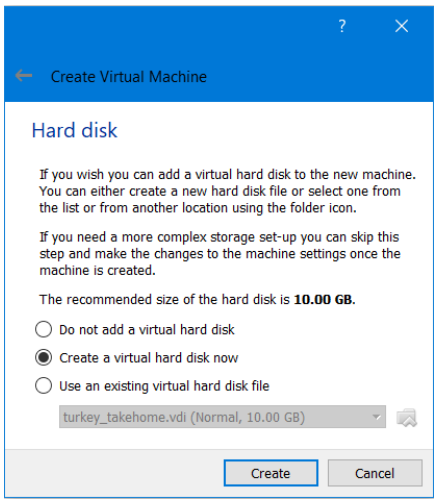
STEP 5: Allocate RAM to the Virtual Machine

The Ubuntu OS recommends at least 2GB of RAM. Allocating anywhere between 4GB and 8GB is enough to use Ubuntu and learn how to use the command line. Then click "Next".

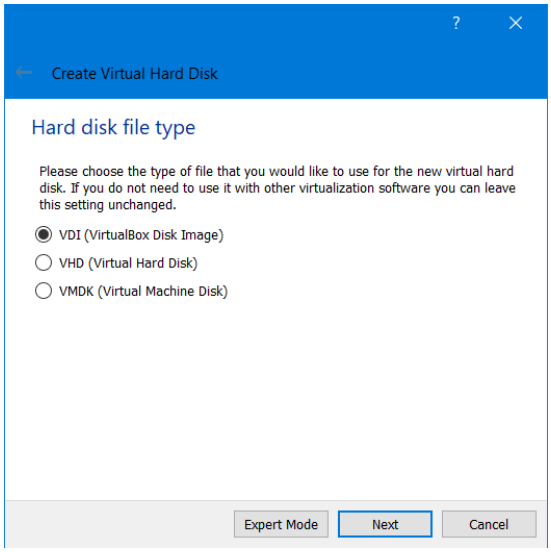


STEP 6: Create the Virtual Hard Disk

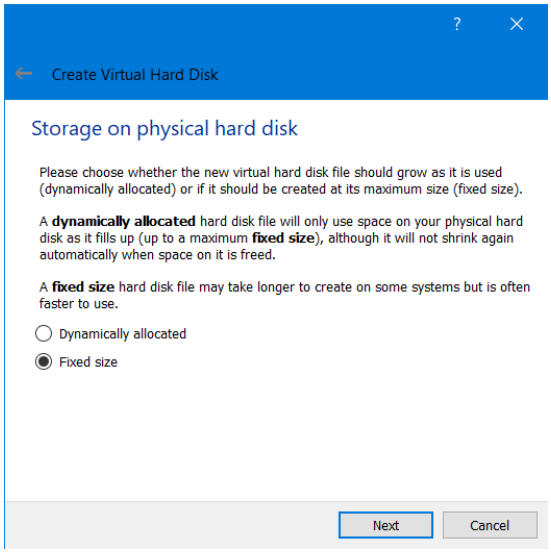
Select the option “Create a virtual hard disk now”, and then click “Create”.



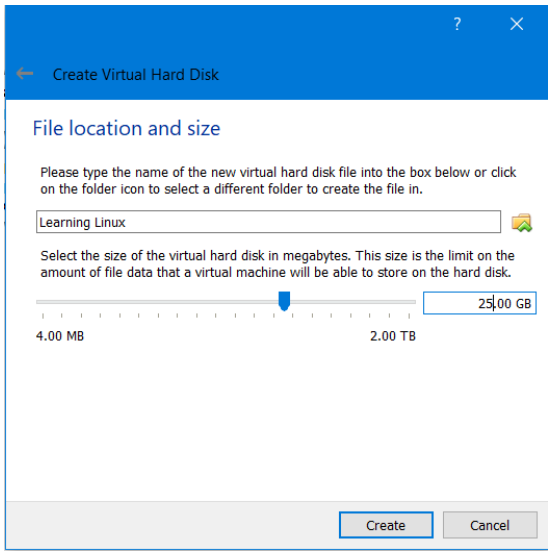
On the next screen, select the VirtualBox Disk Image (VDI) as the hard disk file type, and then click “Next”.



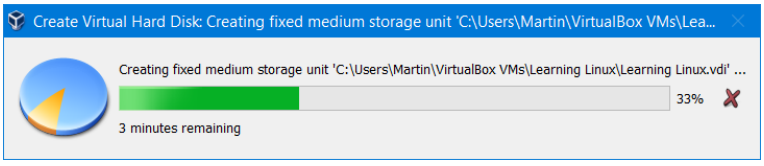
Select the “Fixed size” option, and click “Next”.



Allocate between 10GB and 25GB for the hard drive size, and then click “Create”.



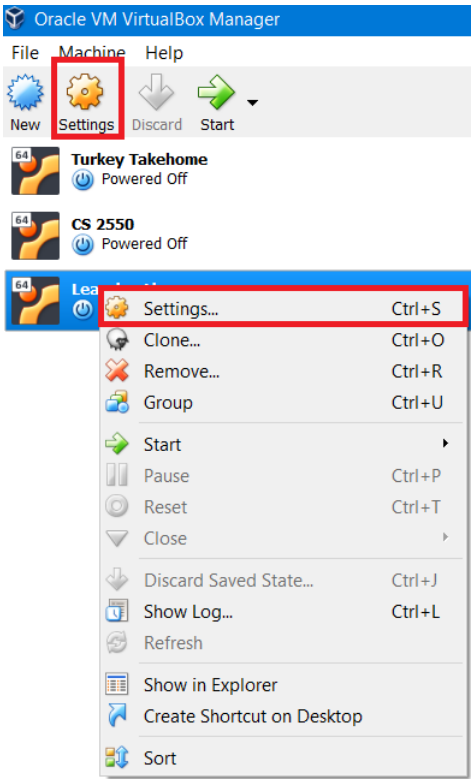
Another screen shows the progress of the creation of the hard drive.



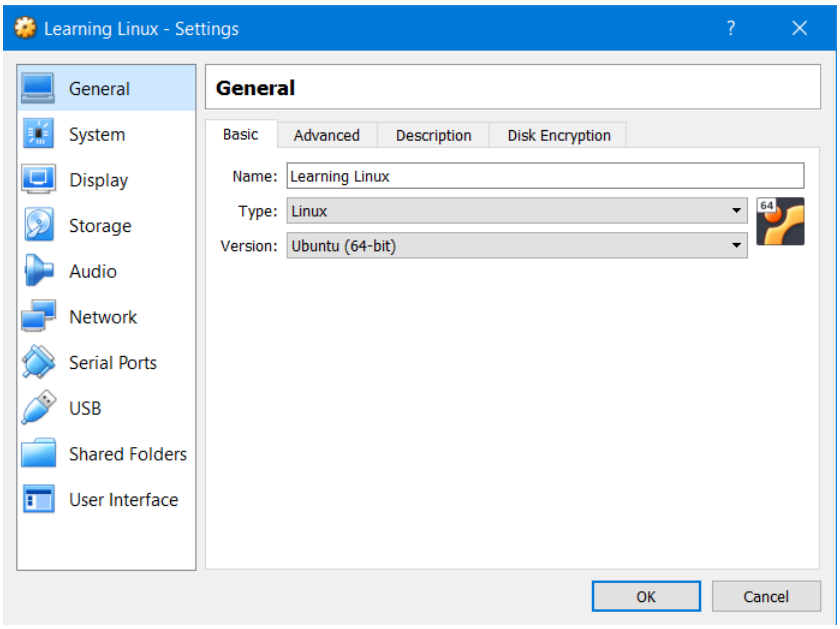
STEP 7: Configure Virtual Machine Settings

Before installing Ubuntu, some settings must be changed for the virtual machine to function properly.

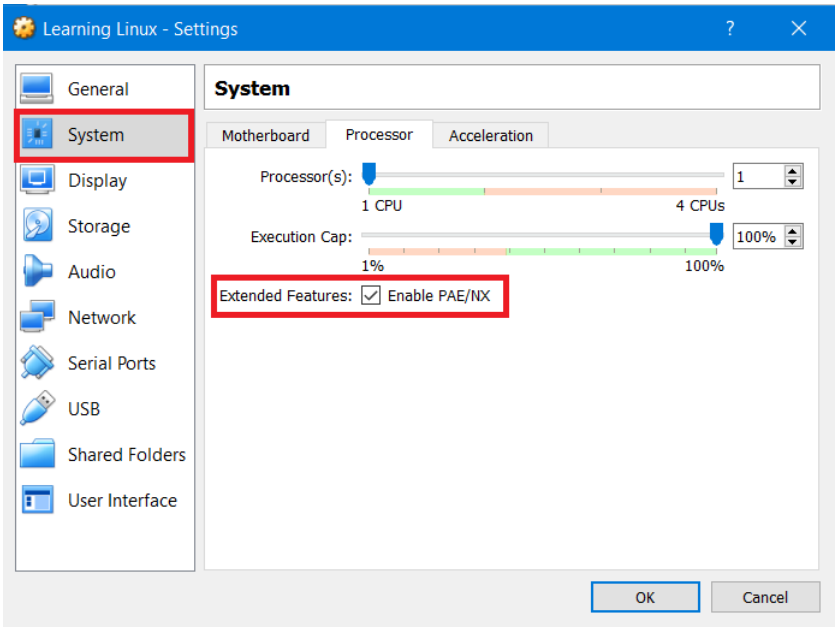
To access the settings of the newly-created virtual machine, either click on the “Settings” button in the top left corner, or right click on the machine and click on “Settings”.



The RAM, number of processors, and other options can be changed from the general settings panel.

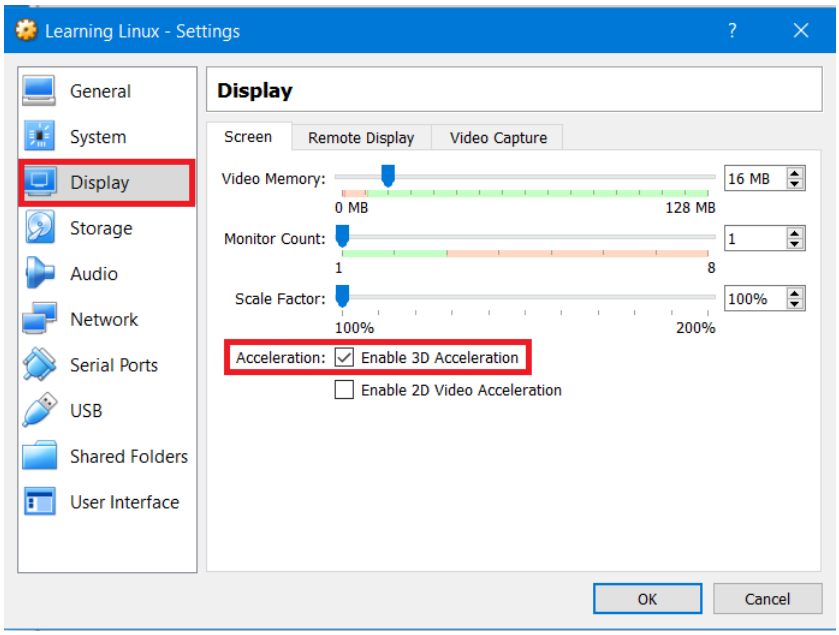


Select “System” on the menu, and then go to the “Processor” tab. Select the option “Enable PAE/NX”.



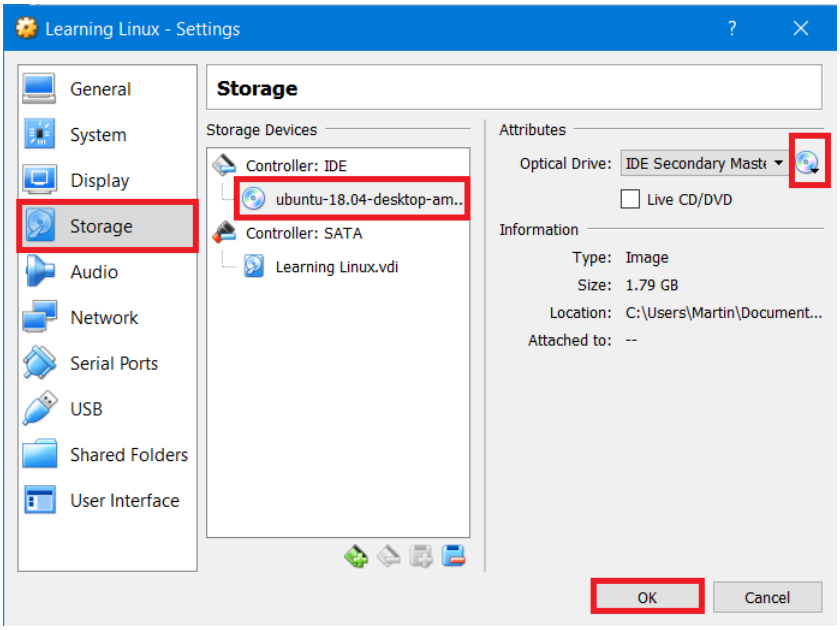
Note: this feature may need to be enabled or disabled for the virtual machine.

Select “Display” on the menu. Under the “Screen” tab, select “Enable 3D Acceleration”.



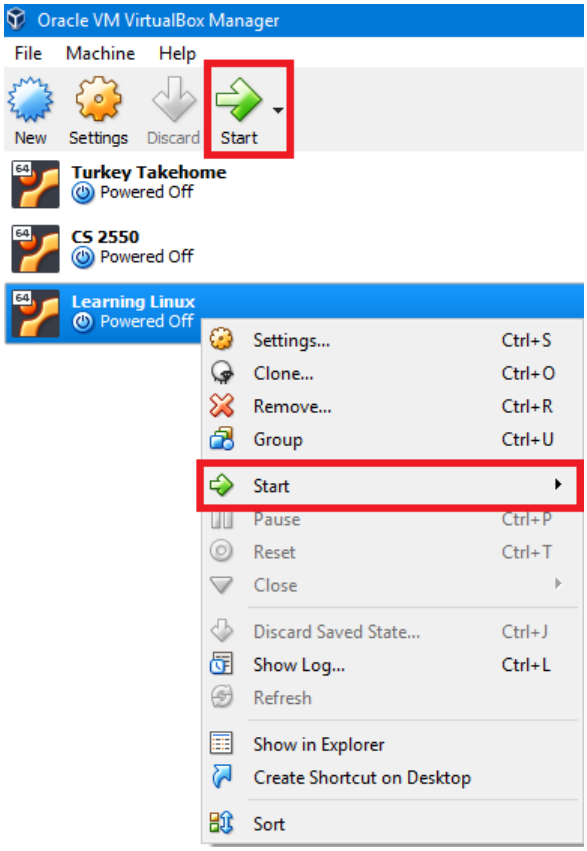
Note: this feature may need to be enabled or disabled for the virtual machine.

Select “Storage” and click on “Empty” under “Controller: IDE”.Then click on the small image of a CD next to “Optical Drive”. A pop up screen will appear to select the Ubuntu ISO file. And then click “OK”.

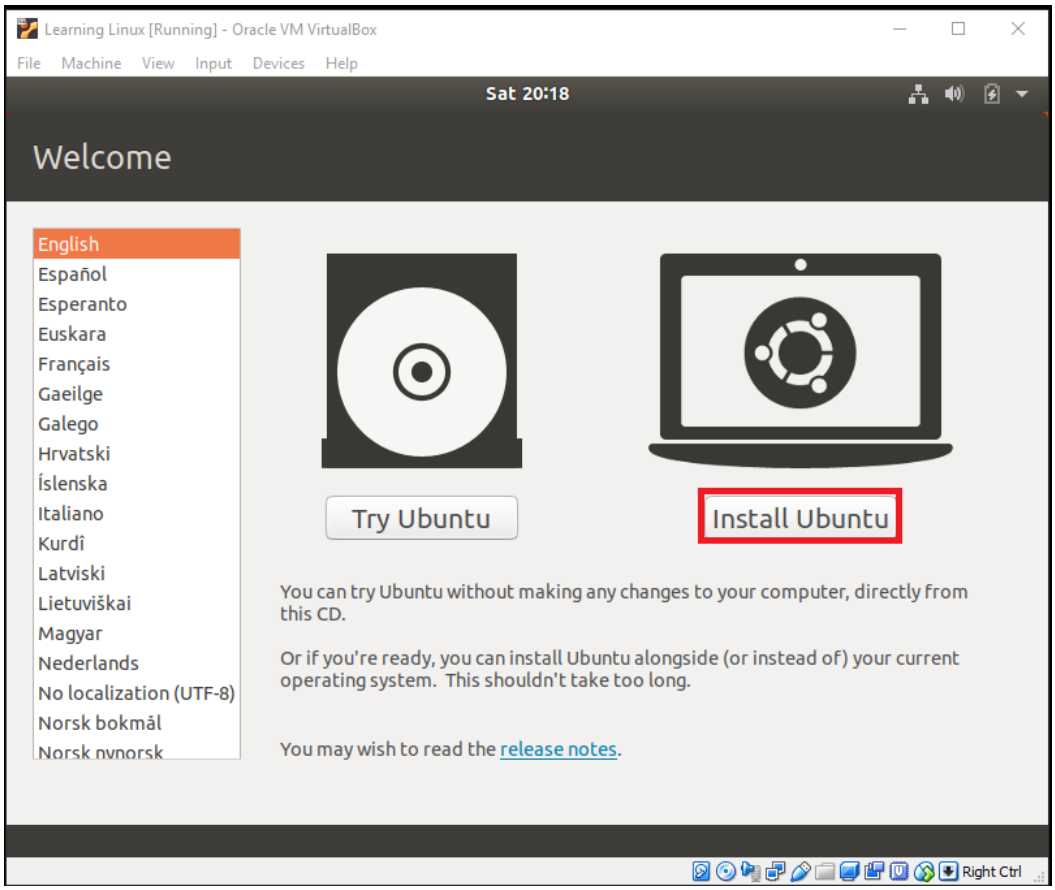


STEP 8: Install Ubuntu on the Virtual Machine

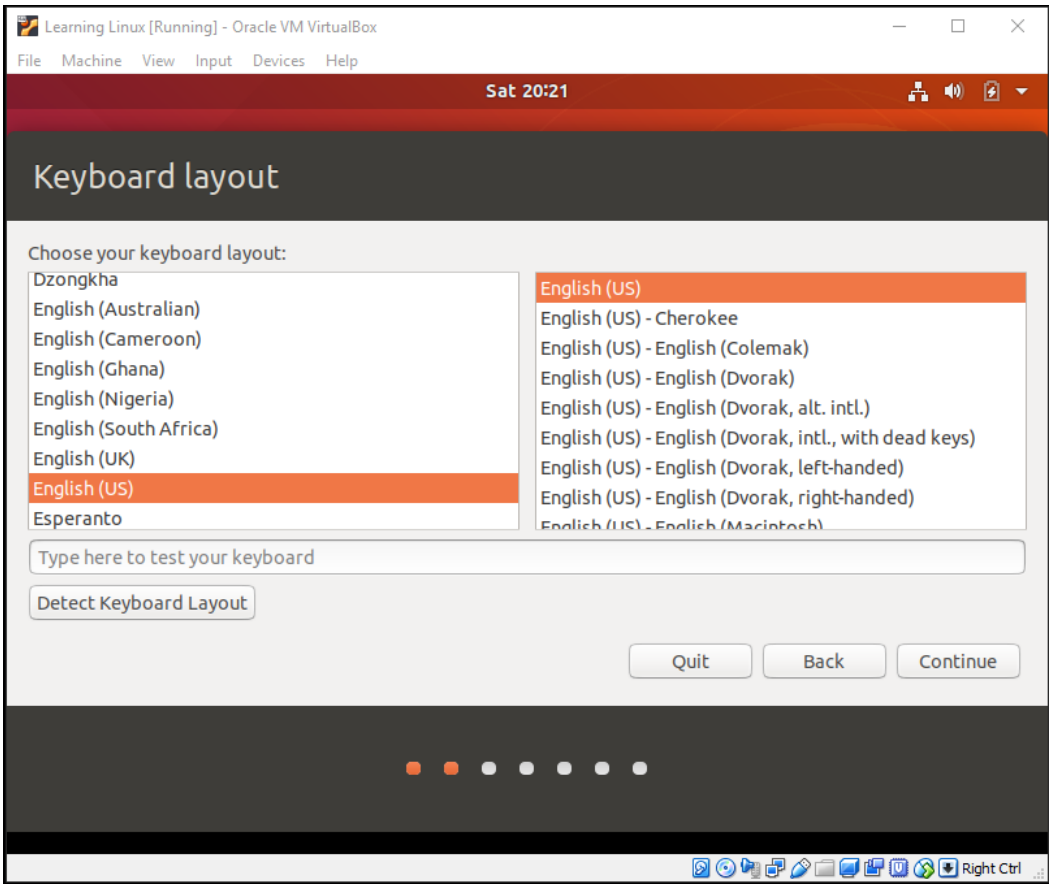
Select the virtual machine that was just created and click “Start”. Either click on the Start button in the upper left corner or right click on the virtual machine and then select the Start option.



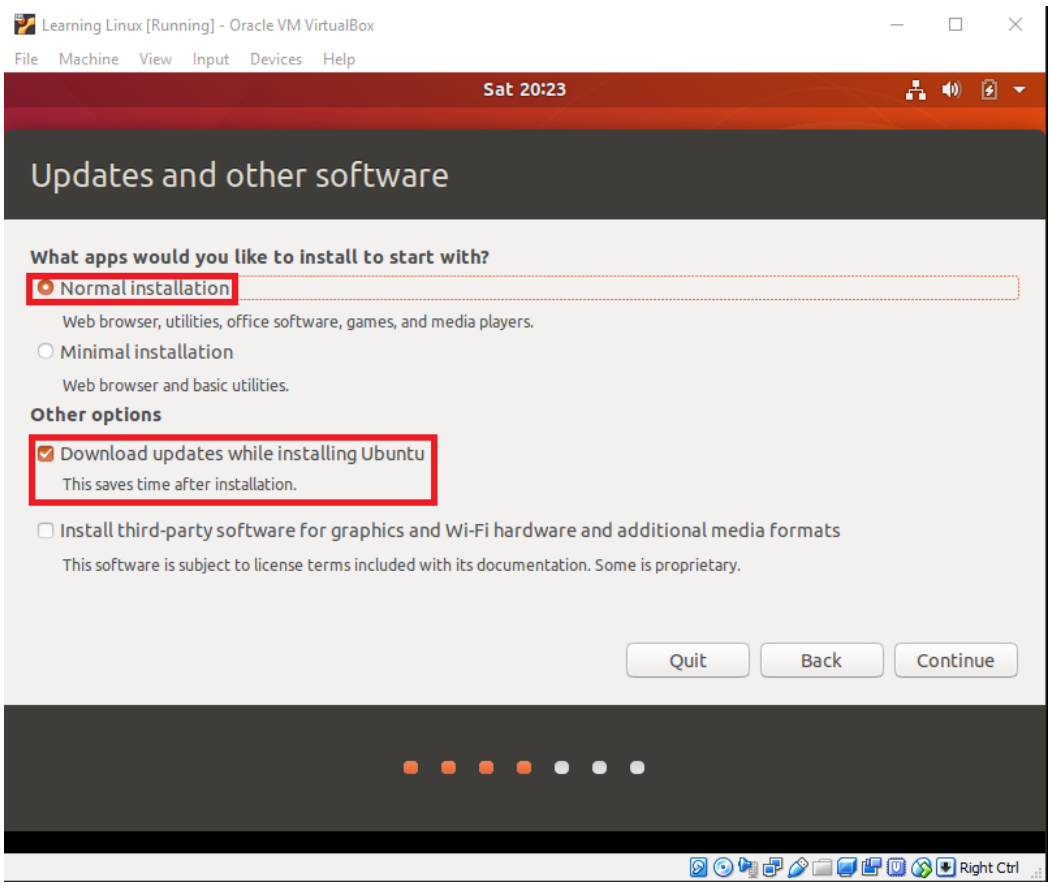
Once the machine is started, select “Install Ubuntu” on the screen that appears.



On the next screen, select the keyboard layout that the Ubuntu OS will use. Then click “Continue”.

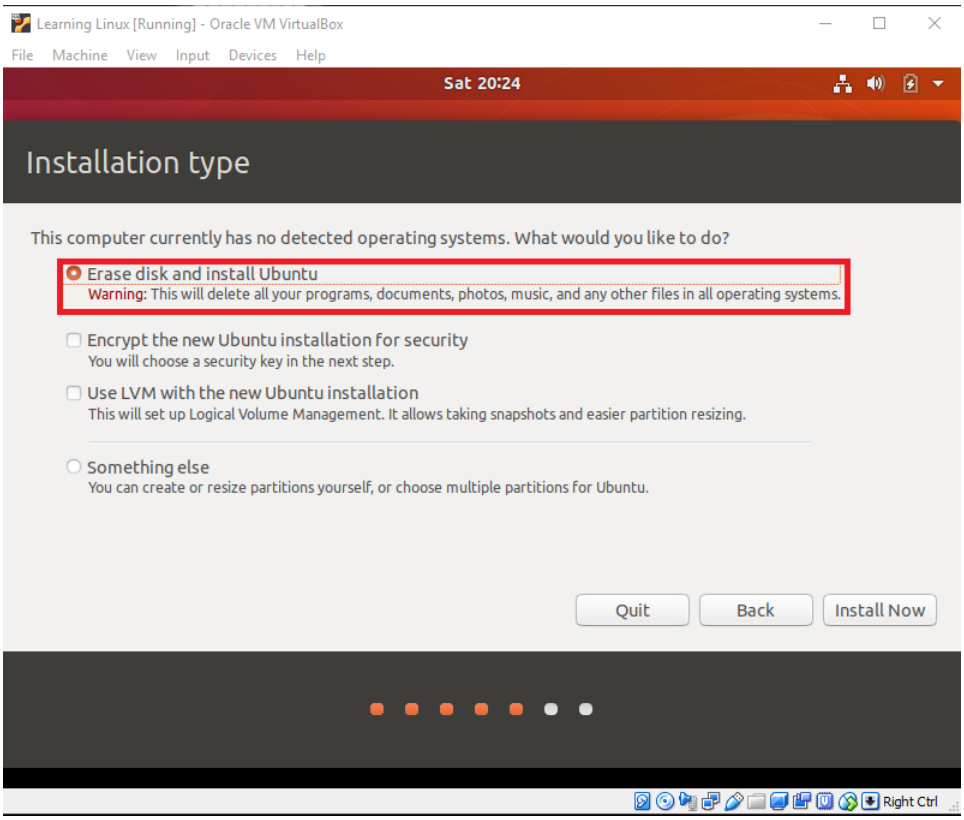


Now select “Normal Installation” to install all the Ubuntu programs, And then select “Download updates while installing Ubuntu”. Then click “Continue”.

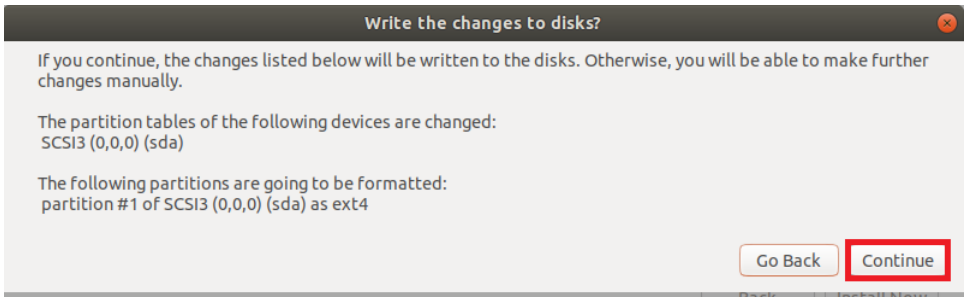


Note: also select the “Install third-party software” if the wifi card or other hardware is incompatible with the virtual machine.

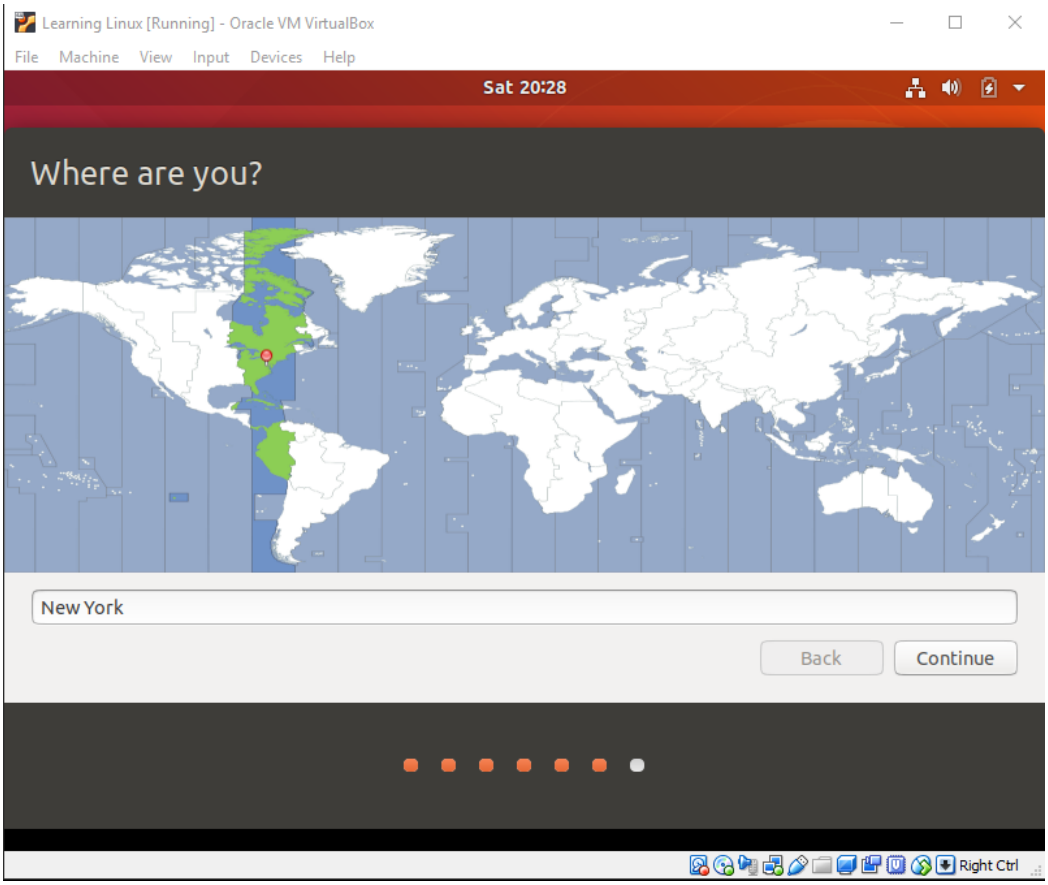
On the installation type screen, select “Erase disk and install Ubuntu” and click on “Install Now”.



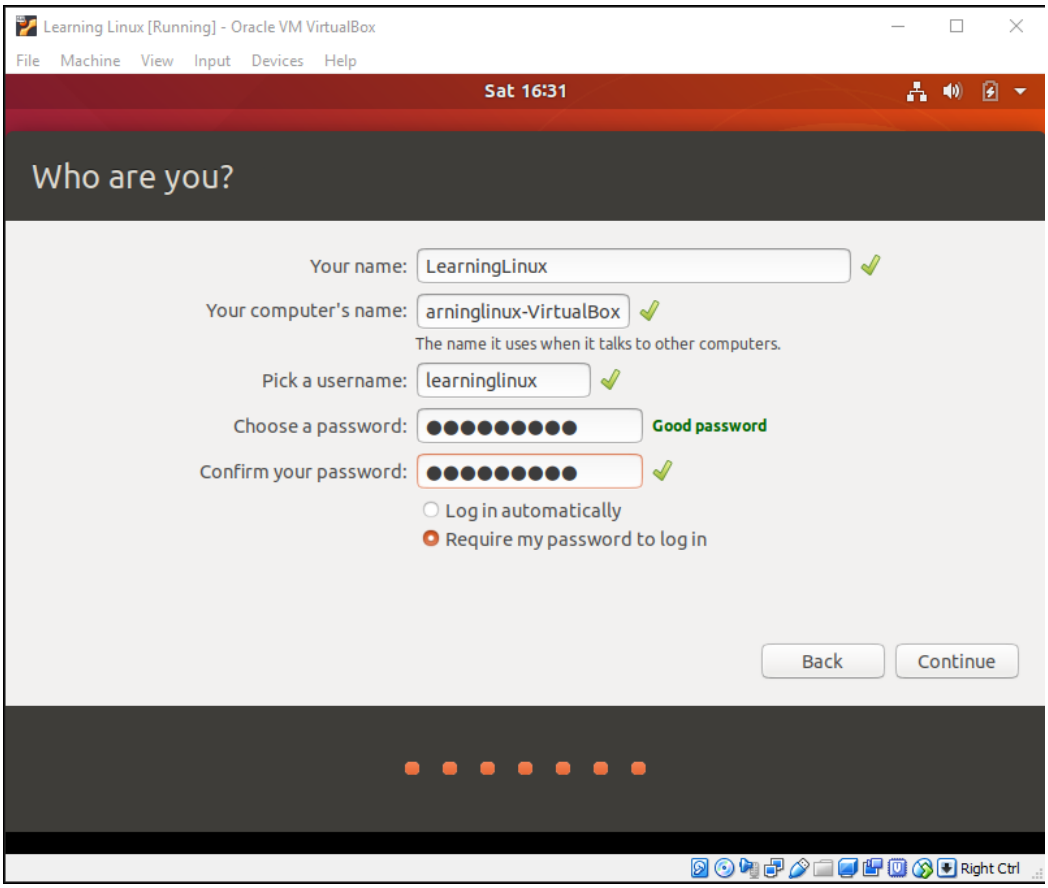
A pop-up appears to confirm the changes. Then click on “Continue”.

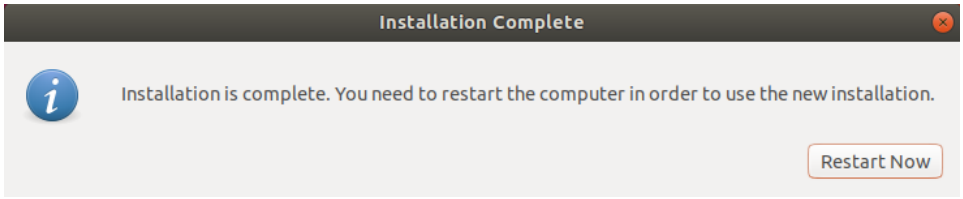


Type in the city to establish the proper timezone. Then click on “Continue”.



Enter the required information in every field. Then, click on “Continue”.





After the installation is completed, click on “Restart Now” to use the virtual machine.

Chapter 2

Basic Navigation of the Command Line

Definitions

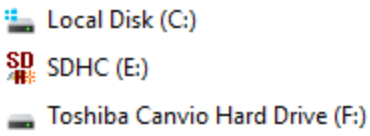
1. **Root Directory** – the top-most directory in a file system structure
2. **Home Directory** – the file system directory for a given user on an operating system
3. **Current working directory** – the directory in which the user is currently in
4. **Absolute path** – the path of a file from the root directory
5. **Relative path** – the path of a file from the current working directory
6. **File system structure** – the organization of the files and directories of a computer

The Basic Navigation of the Command Line

There are many similarities between the file structure of a Windows/Mac OS and the Linux OS. However, some differences, such as the name of the home and root directories, should be noted.

On the Windows OS, the home directory is `C:\Users\<USERNAME>`. This is where user information is stored, such as documents, pictures, etc.. `C:\` is the root directory of the entire computer. It is where the information for the computer is stored in different folders such as program files, operating system files, and other files.

Other devices connected to the computer have different corresponding root directories starting with another letter. Look at the following example of drive letter assignment on a computer.

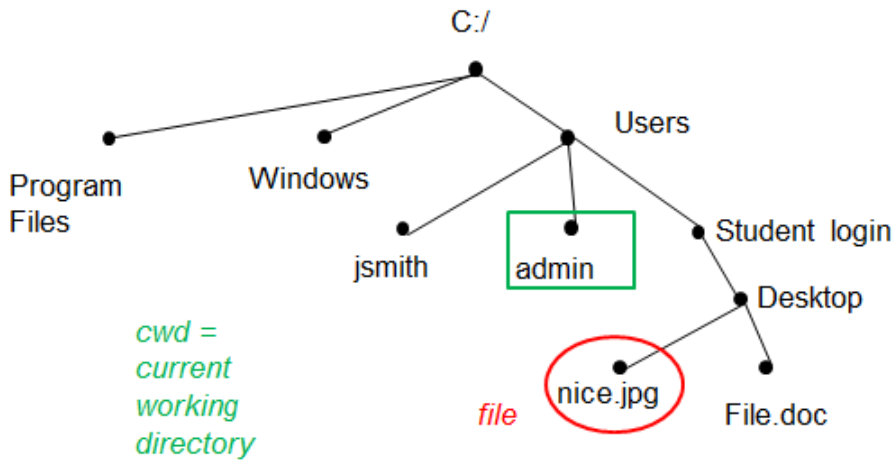


`C:` is the root for the local disk, which is the computer, while `E:` is a SD card and `F:` is an external hard drive. Windows assigns the next free letter to the appropriate device. A file in the external hard drive might have the filepath of `F:\projects\examples1.doc`.

On a Linux-based operating system, the home directory is `/home/<USERNAME>`. Like the Windows OS, that is where all the user information is stored. However, one of the main differences is that the root directory for a Linux machine is `/`, which is a simple forward slash.

The location of other devices in a Linux computer will be explained in the end of this section.

Directory Structure



The above is an example of the directory structure on a Windows computer. The file system is hierarchical and it looks just like a tree. For a Linux machine it would have the same hierarchical structure with different directory names.

To navigate the directories of any operating system, it is necessary to have a basic understanding of the file system structure.

Absolute Paths vs. Relative Paths

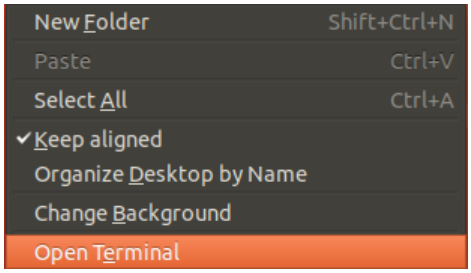
An absolute filepath is the path for a file from the root directory, which in the diagram on p.22 is `C:\` which is at the very top of the tree-like structure. For example, the absolute filepath for the file nice.jpg, which is circled in red in the above diagram, would be `C:\Users\Student login\Desktop\nice.jpg`.

A relative filepath is the path for a file from your current working directory, which, in the diagram on p.22, is the admin folder boxed in green. For example, the relative filepath to the file nice.jpg would be `..\Student login\Desktop\nice.jpg`. Look closely at how the filepath starts. It starts with a two periods. This signifies going up one level in the directory structure from the admin directory to the Users directory.

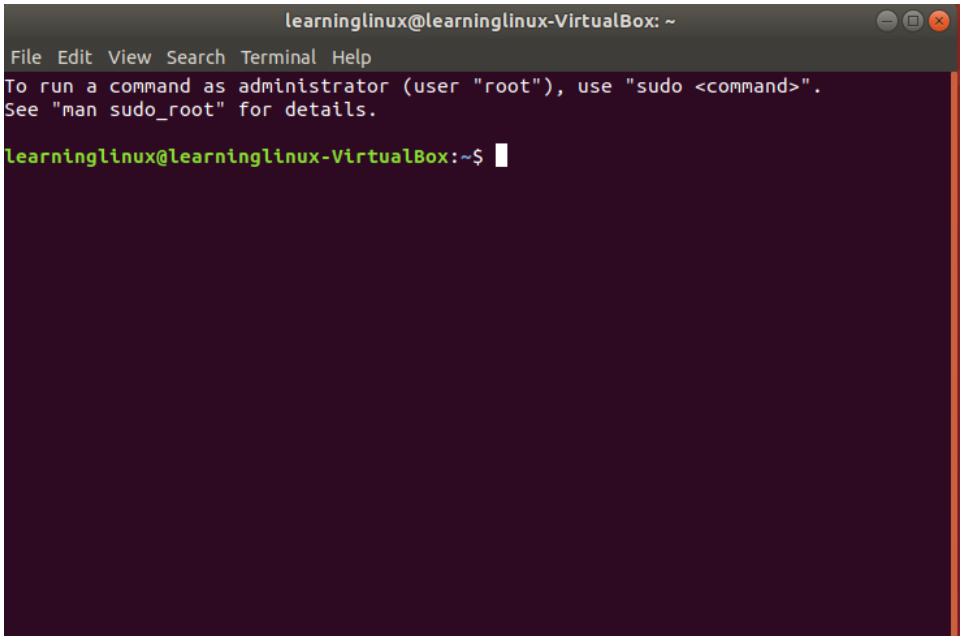
Using the Terminal

Now that you know the basics of directories and filepaths, it is now time to start using the terminal.

To start up the terminal, right-click anywhere on the screen to display the below menu. One of the options is “Open Terminal”.



Selecting “Open Terminal” displays the below terminal, and from there you can start to use the command line.



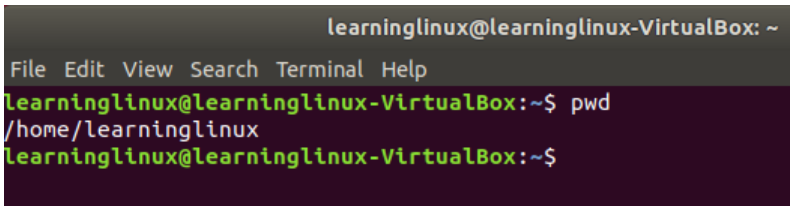
Entering Commands on the Terminal

Within the terminal, you have something called a shell. The shell is a command-line interface that interprets commands, defines how the terminal behaves, and how it looks after executing commands. On most Linux distributions, including Ubuntu, Bash is the default shell. Bash stands for Bourne-Again shell. Whenever you start a terminal, its location is always your home directory.

The terminal displays the following when it first starts up:

1. **learninglinux**: the username of the current user
2. **learninglinux-VirtualBox**: the name of your virtual computer
3. ~: the current user's home directory, which in this case is /home/learninglinux
4. **$**: prompt symbol, your input appears after this symbol

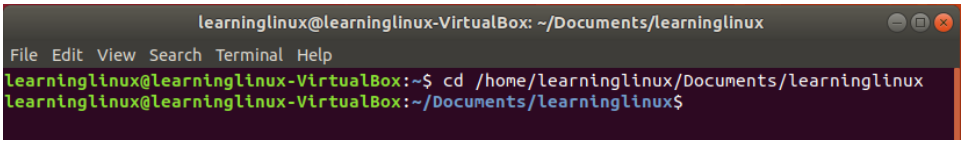
The **pwd** command stands for print working directory. The syntax for the command is as follows: `pwd [OPTIONS]`. Without any arguments, this command prints the absolute file path of the directory that you are currently in. Type *pwd* in your terminal to see the result. It should look similar to the following, but instead of learninglinux, it will display your username.



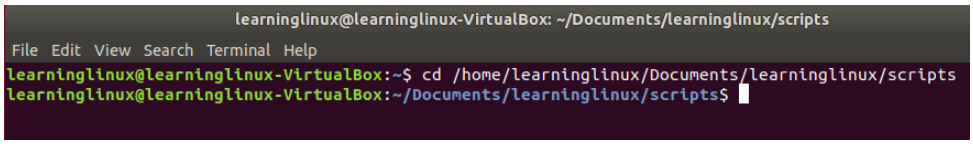
Note: There are optional arguments for the **pwd** command, but you do not need to know them right now

To move to other directories, use the **cd** command, which stands for change directory. The syntax is `cd [DIR]`. DIR is an optional argument for the location of the directory you want to move to. If no argument is supplied, the cd command automatically goes back to the home directory.

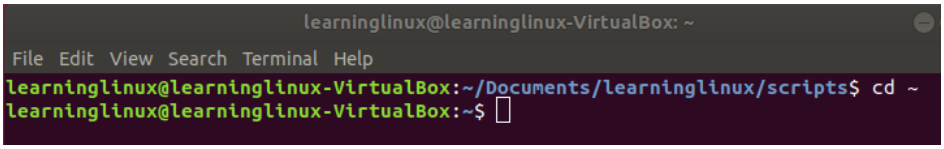
To use the **cd** command to move around, use either the absolute path or the relative path of the directory. There is a directory named learning Linux within my Documents directory. To change to this directory using the absolute path, type the command *cd /home/learninglinux/Documents/learninglinux*.



Another example is to go into the scripts directory, which is within the learninglinux directory. To do this, run the command *cd home/learninglinux/Documents/learninglinux/scripts*.

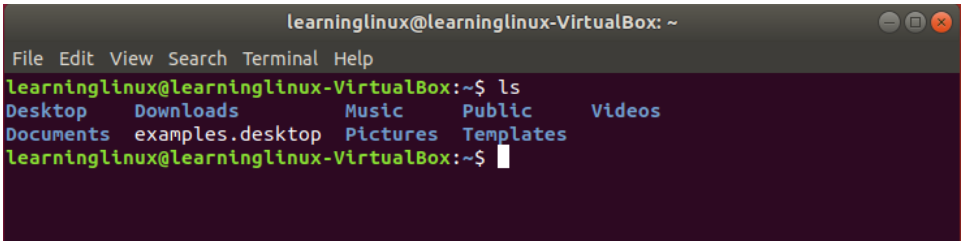


In both of these examples, the shell shows your current working directory, which starts off with `/Documents/learninglinux`. The ~ symbol shows up again and it has the same meaning as we mentioned before. It is a shortcut for the home directory. Now type in *cd ~*. This will change the directory back to the home directory.



Viewing Directory Contents

To know what is in the directories use the **ls** command, which stands for list segments. The syntax for this command is `ls [OPTIONS] [FILE]`. By default, the command with no arguments will list all visible files and directories that are in the current directory. In later sections we will learn some of the other optional arguments. Enter *ls* in your terminal.

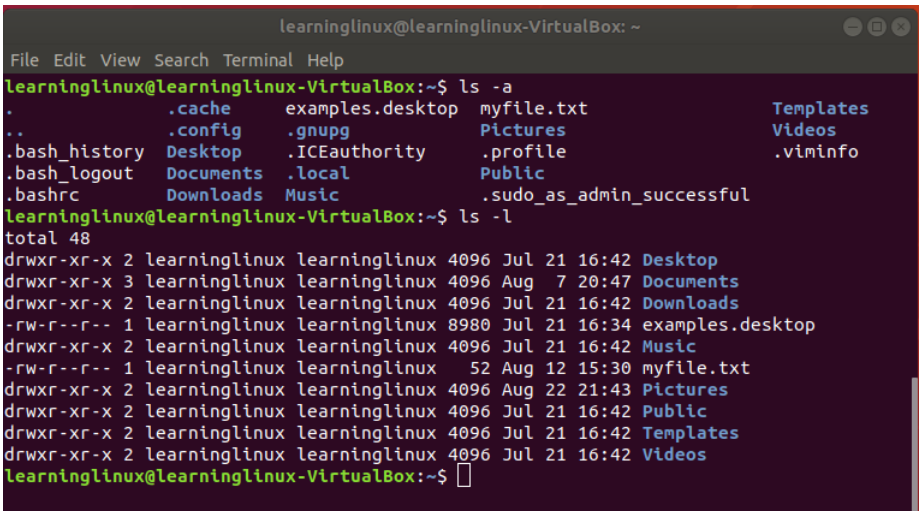


As you can see above the different files and directories are color coded. The following table describes the color and their associated file type.

Color	File Type
White	Text Files
Blue	Directories
Green	Executables
Pink	Images
Red	Archives
Cyan	Links

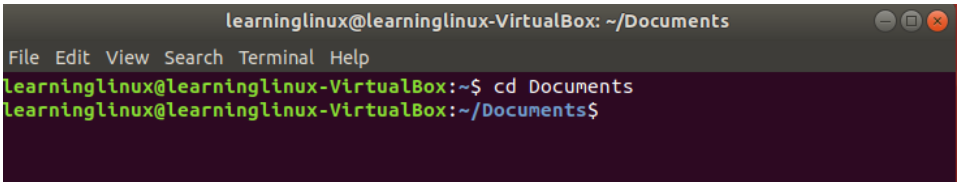
Note: These colors may differ with different versions of the shell.

The **ls** command has numerous optional arguments that can be passed along with the command. Two of the most common arguments with the **ls** command are `a` and `l`. `a` will show all the contents of a directory, including hidden files and `l` will list more information about each item in the directory. Try both of these commands in the terminal.

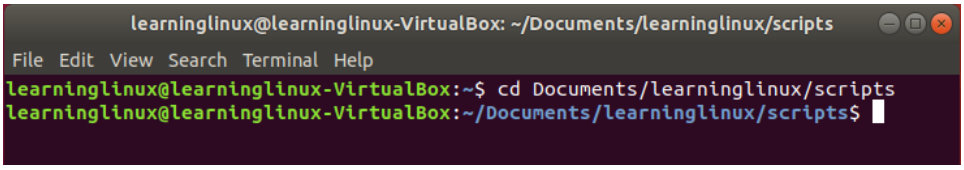


Look closely at the left most column of the output of ls -a. The first two items displayed are a "." and "..". These are very important in the file system structure. Both of these will always be present in a directory, no matter which one it is. The single period represents the current directory you are in, and the double period represents the parent directory.

To move around from one directory to another using relative paths, enter the name of the directory relative to the current working directory. For example, to go from the home directory to the `Documents` directory, type the command *cd Documents*.



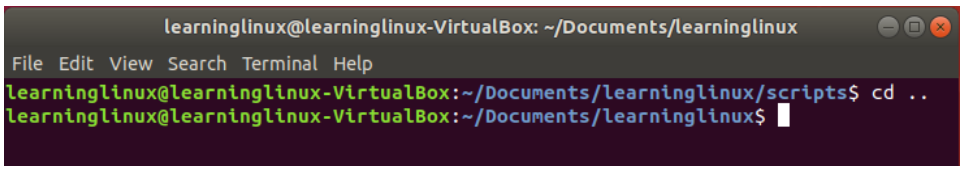
You can also move several directories down at a time. Within the `Documents` directory, there is a directory named `learninglinux`, and within that directory, there is one called `scripts`. To get to that directory in one command, type in the command *cd Documents/learninglinux/scripts*.



Note: You may run into some issues with directory names if they contain spaces or other special characters. These types of issues will be discussed in a Chapter 7 Complex Inputs

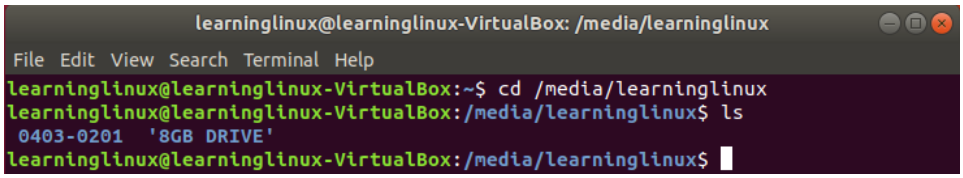
So far, all the commands have been going down into directories. But there are ways to go back up in the file system structure. Right now, the current working directory is `Documents/learninglinux/scripts`.

To get to the `Documents/learninglinux` directory, type in the command *cd ..* The two periods after **cd** signifies the parent directory, which is up one level in the directory structure.



Linux Devices

Other devices on Linux are stored in the `media` directory. Use the command *cd /media/<USERNAME>* to navigate to the directory that may contain other devices. This is yet another example of using the absolute path with the **cd** command.



This directory contains the other devices that are connected to the Linux machine at the moment. There are two: the first one is an SD card, and the second is a flash drive. To see the contents of these drives, change into the appropriate directory using the **cd** command.

Command Summary

Command	Purpose
pwd	print name of current/working directory
cd or cd ~	change the working directory to the home directory
cd [DIR]	change the working directory to DIR
cd ..	change the working directory to the parent directory
ls	list directory contents
ls -a	list directory contents including hidden files
ls -l	list directory contents with details

Chapter 3

File Basics

Definitions

1. **File** – a computer resource for recording data that is accessible by the operating system, programs, and the user
2. **Character Encoding** – a mapping of characters to bytes so that a computer can interpret characters properly
3. **Bit** – the smallest unit of data that can be represented on a computer
4. **Byte** – unit of data comprised of eight bits

This section covers what a file is, what character encoding is, and how to create files from the command line.

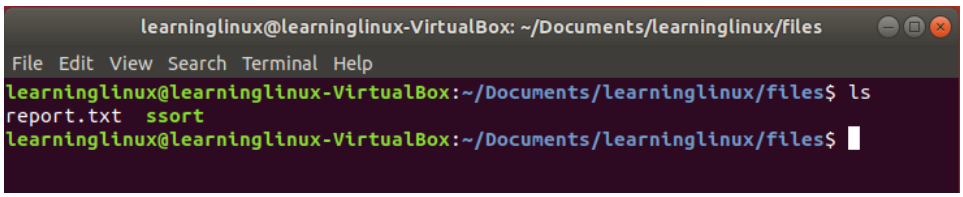
Files

A file is simply a sequence of bytes written on a computer storage device, such as a hard drive, flash drive or other device, that is accessible by the operating system, different programs, and the user. In essence, files are an abstraction for organizing groups of bytes that go together. For example, a Word document might contain text, a table, a couple charts, and some meta-data about how to format all of the before-mentioned content. All of these bytes are stored together in a single file since we consider them all to be part of the same document.

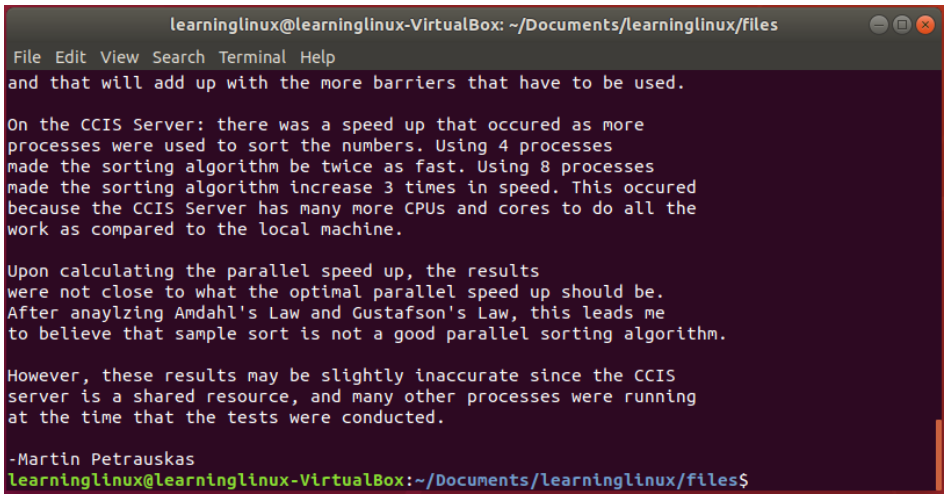
Text and Binary Files

There is a distinction to be made between text files and binary files. Text files contain human readable text while binary files contain data that cannot be interpreted by a human, and only by a computer program.

The result of running the **ls** command in the `files` directory is two files. One regular text file, and binary file.



To see the information in the file, use the **cat** command, which stands for concatenation. The syntax for this command is `cat [OPTION] [FILE]`. The result of running the *cat report.txt* on the regular text file is the following.



As you can see, it is definitely human readable text.

Now, if we run *cat ssort*, we will not see any human readable text.

Other than the occasional character that can be interpreted, the output of the binary is unreadable.

Character Encoding

Computers can only interpret bits, which store binary values of 0 or 1. This means that they cannot interpret characters like 'a', '4', or '%'. These characters are stored in the computer as one or more bytes. As a result, a character encoding maps computer bytes and the characters a person sees on the screen.

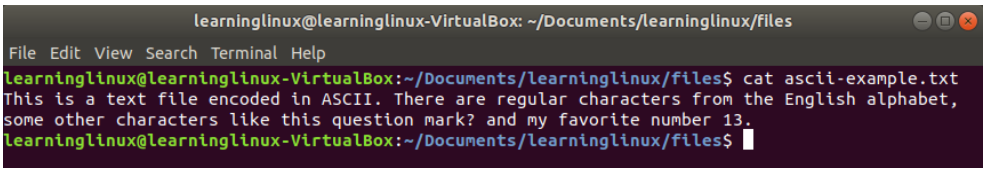
ASCII - American Standard Code for Information Interchange

ASCII is the oldest standard for character encoding. All ASCII characters are encoded using one byte of information (8 bits). However, this limits the number of characters that can be encoded to 2^8, which is 256 characters, such as the English alphabet, roman numerals, and some special characters.

The following is a table of characters encoded with ASCII:

Decimal	Binary	Character
0	00000000	NULL
42	00101010	*
77	001001101	M
13	00001101	CR (carriage return)
7	00000111	BELL

And the following is an example of a file encoded in ASCII.



UTF - Unicode Transformation Format

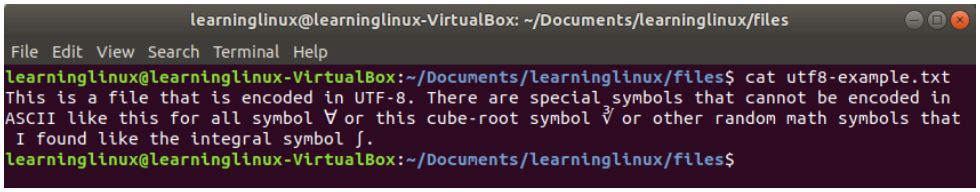
ASCII was a good start for encoding characters, but what about all the other characters that are found in different languages? Those could not be represented with the number of characters that ASCII is limited to.

In 1987, people from Xerox and Apple set out to make a universal character set that would include characters from all the different languages of the world. The major character encodings in UTF are UTF-8, UTF-16, and UTF-32.

UTF-8 is the most common encoding and uses four bytes to encode a character. It was designed to be backwards compatible with ASCII. This means that the first byte of information in UTF-8 has the same character mappings as ASCII.

Because of this backwards compatibility, whenever you are coding or typing a file, it will default to an ASCII text file, but as soon as it encounters a non-ASCII character, the file will then be encoded in UTF-8.

Here is an example of a file encoded in UTF-8.



Line Endings

Any machine that runs DOS/Windows uses different line endings than a machine that runs Unix. On Windows machines line endings are "\r\n", which is the carriage return followed by a newline. However, in Unix the line endings in just the newline represented by "\n".

There is a command that can be used to convert the file endings from Dos style to Unix style and this program is called **dos2unix**. The syntax for this command is `dos2unix FILE`, which would convert all the line endings in the `FILE` to the proper Unix-style line endings.

File Extensions

File extensions are the endings attached to the names of files, such as exe (executable), jpg (picture), doc (document) and many others. These extensions signal to the operating system how the files should be interpreted and how the bytes of information are arranged.

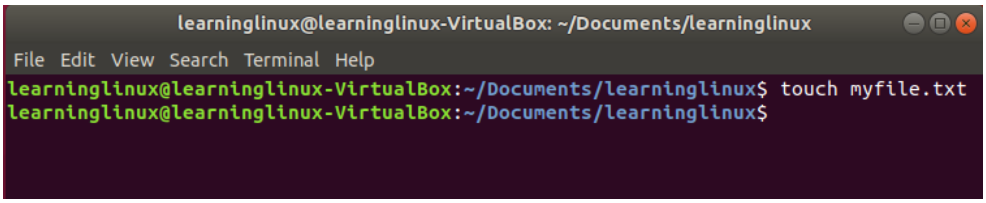
Operating systems rely on file extensions to find the proper program to open a file. It would be bad if Notepad++ tried to open a mp3 file. This is very common in graphical interfaces when a user can double click on a file, and the corresponding program will open the file.

However, command lines do not care about file extensions, and for the most part, they will not associate a file with any type of program. There is a small exception on Windows OS where the command line will assume a file is executable if it has the extensions exe. This issue does not arise in Linux because file permissions denote if a file is executable. File permissions will be explained in Chapter 5.

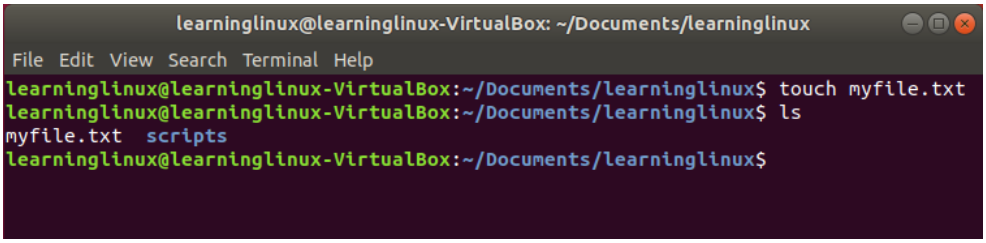
Creating a New File

To create a file from the command line, there are several commands that can be used.

If you want to create an empty file, use the **touch** command. The syntax for the command is `touch [OPTION] [FILE]`. Go to the terminal and type *touch myfile.txt*.



This creates a text file called `myfile.txt` in your current directory. To check that the file is there, enter the *ls* command.



Another way to create a new file is to use a command line text editor such as **vim** or **emacs**. The synatx for these commands are `vim [FILE]` or `emacs [FILE]` and these a text editor will open up on the command line so that content can be added.

Command Summary

Command	Purpose
touch [FILENAME]	creates an empty file named FILENAME
cat [FILENAME]	displays the contents of FILENAME
vim [FILE]	opens the FILE with the vim text editor
emacs [FILE]	opens the FILE with the emacs text editors
dos2unix [FILE]	converts the FILE to Unix-style line endings

Chapter 4

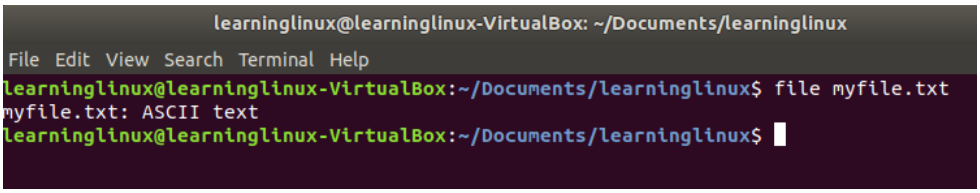
File Manipulation

In this section, you will learn about files names, creating, moving, copying, and deleting directories and files.

File Names In Linux

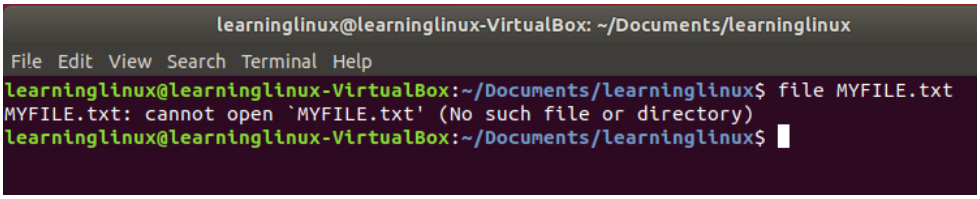
File names in Linux are case sensitive. That means the name of the file is specific to the uppercase and lowercase characters. For example, if a file is named `myfile.txt`, and we try to open a file named `MYFILE.txt`, an error message will appear. These are two distinct files.

To determine a file type, use the command **file**. The syntax for this command is `file [FILENAME]`. At the terminal, type in *file myfile.txt*.

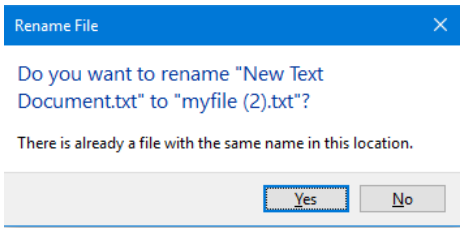


The output is the name of the file, which is `myfile.txt`, followed by the type of file, which is a basic text file encoded in ASCII.

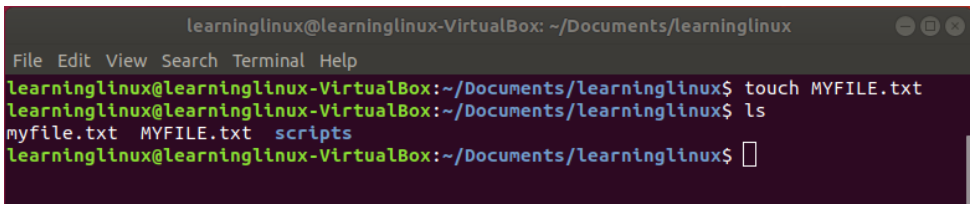
Try the command *file MYFILE.txt* and an error message appears that the file does not exist.



In the Windows and Mac OS, `myfile.txt` and `MYFILE.txt` would refer to the same file. If you have a file named `myfile.txt` and you try to create a file named `MYFILE.txt`, you will see the following error message in Windows.

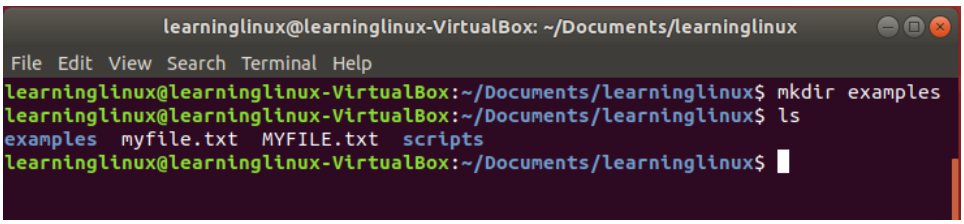


These operating systems regard the files as the same even though their case is different. In the following figure, you can see that I created a file named `MYFILE.txt` and Linux does not display an error message. Looking at the directory contents, you can see two distinct files.



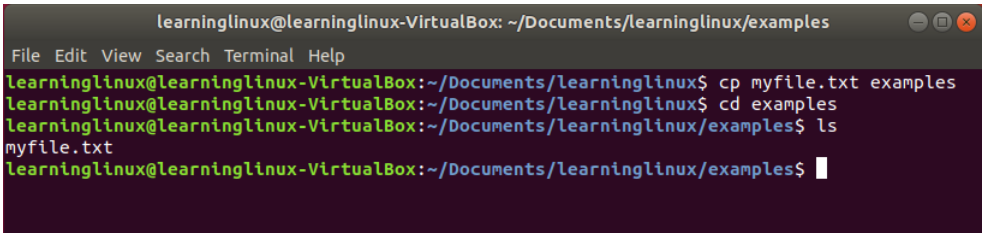
Creating Directories

To make directories from the command line, use the command **mkdir**, which is shorthand for make directory. The syntax for the command is `mkdir [OPTIONS] DIRECTORY`. At the terminal, type in *mkdir examples*. This will create a directory named examples.

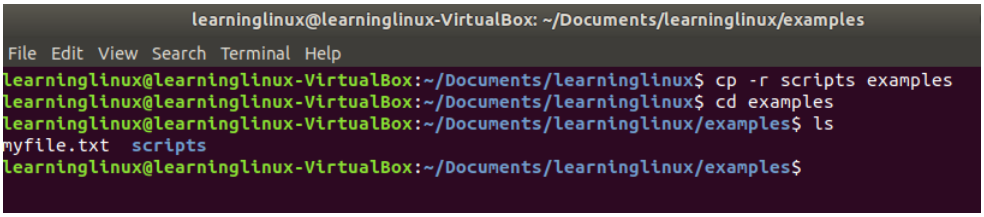


Copying Files and Directories

To copy files and directories, use the **cp** command, which is shorthand for copy. The syntax for the command is `cp [OPTIONS] SOURCE DEST`, where `SOURCE` is the file or directory you want to copy and `DEST` is the destination The command to copy `myfile.txt` from the current directory to the newly created examples directory is *cp myfile.txt examples*.



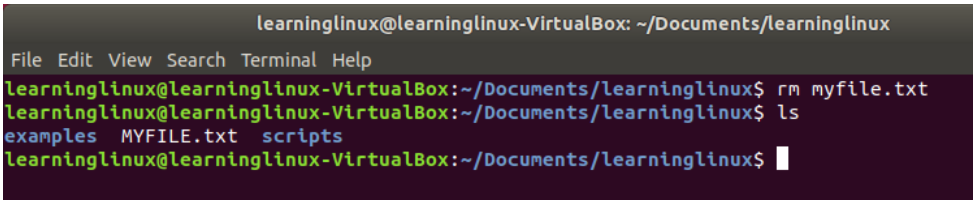
Additionally, an entire directory can be copied instead of one file. The command to copy the scripts directory into the examples directory is *cp -r scripts examples*. In this case, the optional argument `r` is required to recursively copies all the contents of the scripts directory into the examples directory.



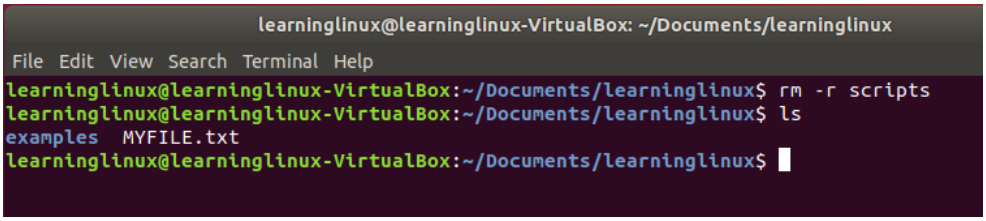
Deleting Files and Directories

To remove files or directories, use the **rm** command, which is shorthand for remove. The syntax for the command is `rm [OPTIONS] [FILE]`.

Navigate to the directory where you want to remove a file. Run the command *rm myfile.txt*. This will delete the file named myfile.txt.



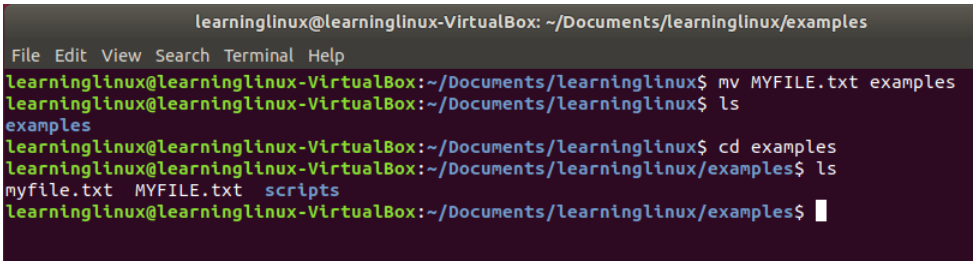
You can also use the **rm** command to delete an entire directory. To delete the directory `scripts`, go to the directory which contains scripts and run the command *rm -r scripts*. The `r` is an additional argument that removes directory contents recursively.



Note: Remember to include the `r` argument when making changes to entire directories. An error message will be shown if this argument is not included.

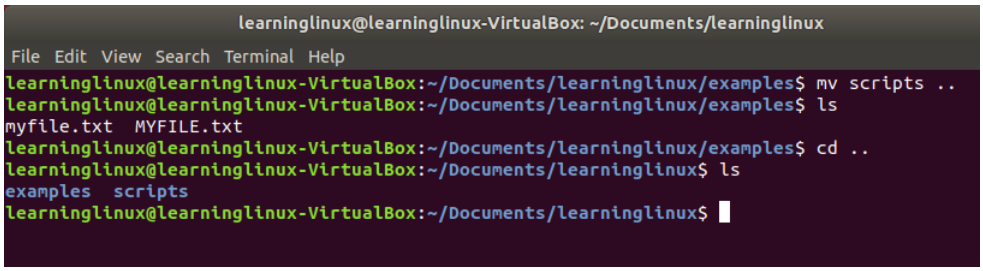
Moving Files and Directories

To move files and directories, use the command **mv**, which is shorthand for move. The syntax for this command is `mv [OPTION] SOURCE DIRECTORY`, where `SOURCE` is the file or directory to be moved, and `DIRECORY` is the destination of the source. To move `MYFILE.txt` from the current directory to the examples directory, use the command *mv MYFILE.txt examples*.



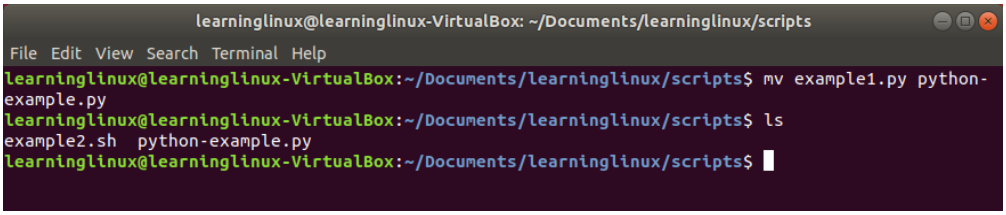
The **mv** command can also be used to move entire directories. To move the scripts directory out of the examples directory up one level, use the command *mv scripts ..*

Note: The two periods mean you are going up one level in the directory structure



Renaming Files

The **mv** command can also be used to rename files. In the `scripts` directory, there is a file named `example1.py`. To rename the file, type in the command *mv example1.py python-example.py*. This will rename the file `example1.py` to `python-example.py`.



Command Summary

Command	Purpose
file [FILENAME]	Determines the file type of FILENAME
mkdir [DIRECTORY]	Make the directory DIRECTORY
cp [SOURCE] [DEST]	Copy the SOURCE TO DEST
cp -r [SOURCE] [DEST]	Copy the SOURCE directory recursively to the DEST
rm [FILENAME]	Remove or delete FILENAME
rm -r [DIRECTORY]	Remove or delete DIRECTORY recrusively
mv [SOURCE] [DIRECTORY]	Move the SOURCE file or directory to DIRECTORY
mv [SOURCE] [DEST]	Rename the SOURCE file or directory to DEST

Chapter 5

File Permissions

Definitions

1. **File Permissions** - rights to access given files for specific users or groups
2. **Octal** - numeral system in base 8 comprised of the numbers 0 to 7

With the basic knowledge about files and how to manipulate them, the next step is to learn about file permissions. Every file has certain permissions that dictate which users can interact with the file and what type of interactions they can have.

Type of File Permissions

Files are assigned any three of the following permissions:

1. **R**ead: allows a user to view the contents of a given file
2. **W**rite: allows a user to modify the contents of a given file
3. e**X**ecute: allows a user to run or execute a given file as a script or program

Every file has a combination of the three permissions. For example, let's say we create the file `myfile.txt`, this file could have read and write permissions, denoted by **rw-**. This means a user can read and write to the file, but cannot execute it. Another file named `script.sh` can have read-write-execute access, which is denoted by **rwx**. This means a user can read, write-to, and execute the file.

With the Linux operating system, files have Unix-style permissions. This means that file permissions are based on three categories:

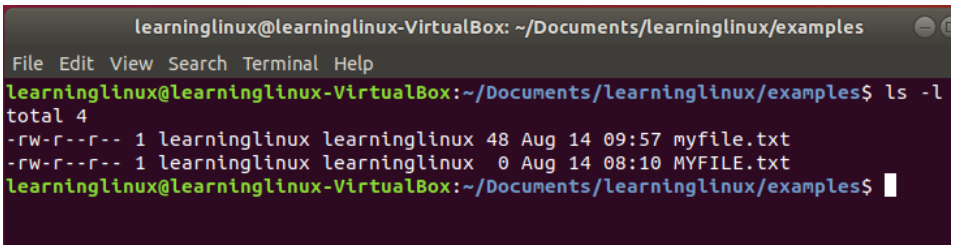
1. Owner/user (**u**): the user who created/owns the file

2. Group (**g**): a specific group of users
3. Other (**o**): all other users

Note: The character in parentheses will be important later on when assigning permissions to specific categories

Reading File Permissions

Navigate to a directory that contains files. To view the permissions of the files in a directory, use the **ls** command with the additional argument `l`. This argument displays the contents of the directory in long list format.. At the terminal, enter the command *ls -l*.



The very first line of the output is "total 4". This is not relevant to file permissions, but it represents the total number of file system blocks all the files take up in this directory.

The next line contains information regarding `myfile.txt`.

1. **-rw-r–r–**: file permissions for myfile.txt
2. **learninglinux**: owner of the file
3. **learninglinux** (the second one): group
4. **48**: size of the file in bytes
5. **Aug 14 09:57**: date that the file was either created either or last modified
6. **myfile.txt**: name of the file

The file permissions have four distinct parts:

1. **-**: The first part is either a simple dash or the character `d`. The dash represents that it is a file, while a `d` represents a directory.
2. **rw-**: The next three characters are for the owner/user of the file. The owner can read and write to this file, but not execute it.
3. **r–**: The next set of three characters is for the group. Anyone in this group has permission to read the files only.

4. **r–**: The last set of three characters is for all other users. They can only read the file.

Another example demonstrates the different groups that could have access to a file or directory.

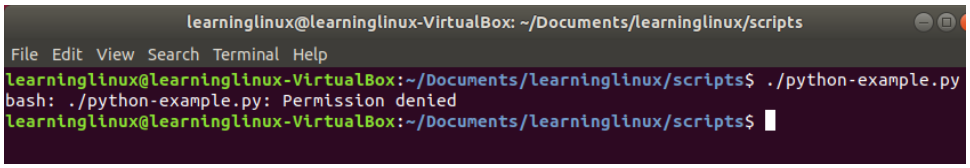
```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

In the first and second listing, the owner and group are the same: "cbw". However, with the third file, the owner is "cbw", but the group is "faculty". Any user that is part of the "faculty" group will have read-write-access to the file. The last file has a special set of permissions. Only the root user has access to this file.

Changing File Permissions - Basic

Changing file permissions allows you to either add or remove permissions from certain files.

There is a file in the scripts directory named `python-example.py`. Try to run this script from the terminal. Type *./example-python.py*. An error message appears and denies permission to run the script.



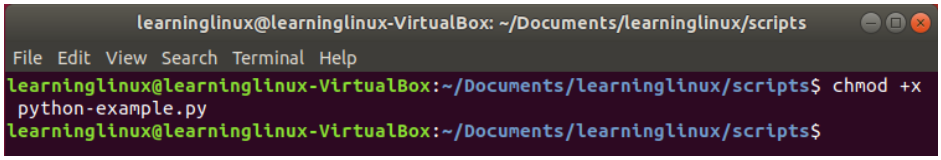
Now take a look at the file permissions for the script. Type *ls -l python-example.py* in the terminal. Adding the filename as an optional argument to the **ls** command applies the command to only the specified file instead of the entire directory.

The file permissions for the python script are: `-rw-r-r-`. This file has no execute permissions. That is why permission was denied.

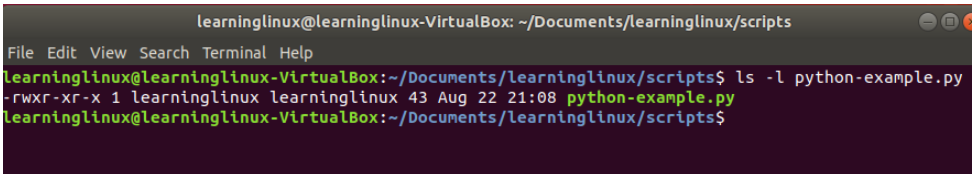
To change file permissions, use the **chmod** command, which stands for change mode. There are multiple syntaxes for the command:

1. `chmod [OPTION] [MODE] [FILE]`
2. `chmod [OPTION] [OCTAL-MODE] [FILE]`

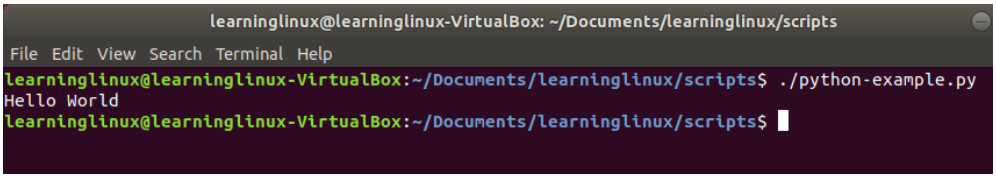
Type *chmod +x python-example.py*. The `+x` means to add permission to execute the file.



Look at the file permissions again using the **ls** command.



Now the file permissions for the script are: `-rwx-r-xr-x`. All groups may now run this script. Run the program to see what it does. Type *./python-example.py*. It is a simple Hello World program.



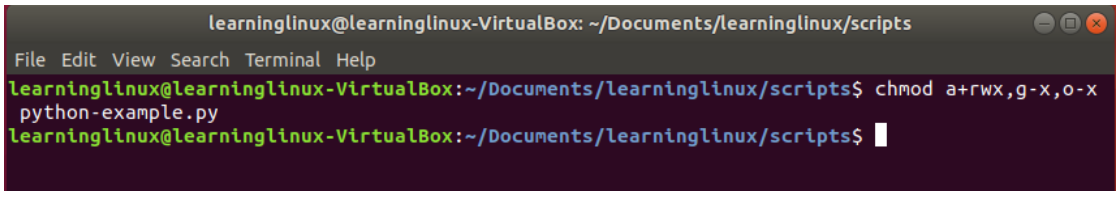
Changing File Permissions - Advanced

In the last example, adding the `+x` argument to the command added executable file permissions to all the different categories (owner, group, and other). That may not be desirable if the file permissions need to be restricted.

Type in the command *chmod a+rwx,g-x,o-x python-example.py*. Here is a breakdown of the command:

1. **a+rwx**: Give read-write-execute permissions to all three categories
2. **g-x**: Remove execute permissions from the group
3. **o-x**: Remove execute permissions from all other users

Now look at the permissions again using the **ls** command.



The file permission is now `-rwx-rw-rw`. Only the owner of the file has the ability to execute the file.

Changing File Permissions - Octal

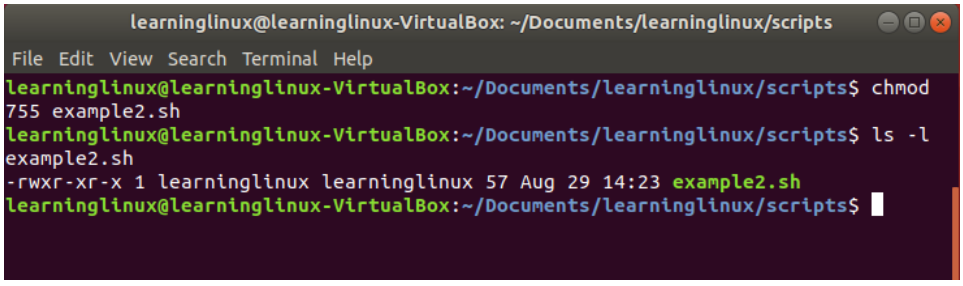
Another way to change file permissions is by using octal mode. Octal mode uses the octal numeral system, which is comprised of the numbers 0-7. Here is a breakdown of the octal numbers for file permissions.

1. **4**: gives read permissions
2. **2**: gives write permissions
3. **1**: gives execute permissions

As noted earlier, there are three categories: owner, group, and other users. Each of these categories has an assigned octal number. This means a three digit number assigns file permissions. The first number is for the owner, the second is for the group, and the last is for other users. To determine the number used for each category, sum the numbers of the associated file permissions. So if you want to give a category read-write-execute permissions, it would be $4 + 2 + 1 = 7$.

In the last example we gave read-write-execute permissions to the owner, and only read-write permissions to the group and other users. In octal mode, this would be 766, and the command would be *chmod 766 python-example.py*.

In the `scripts` directory, there is a script named `example2.sh`. This is a shell script that does not have an execute permission at the moment. Additionally, only the owner should be able to write to the script. Every other category can read and execute the script. With these criteria, the proper octal would be 755. Go to the terminal and type *chmod 755 example2.sh*.



As a result of the command, the file permissions changed to `-rwxr-xr-x`, which is the desired outcome. To learn the different combinations of octal numbers for the chmod command, go to http://chmodcommand.com. This is a helpful website for beginners to learn chmod octal mode.

Command Summary

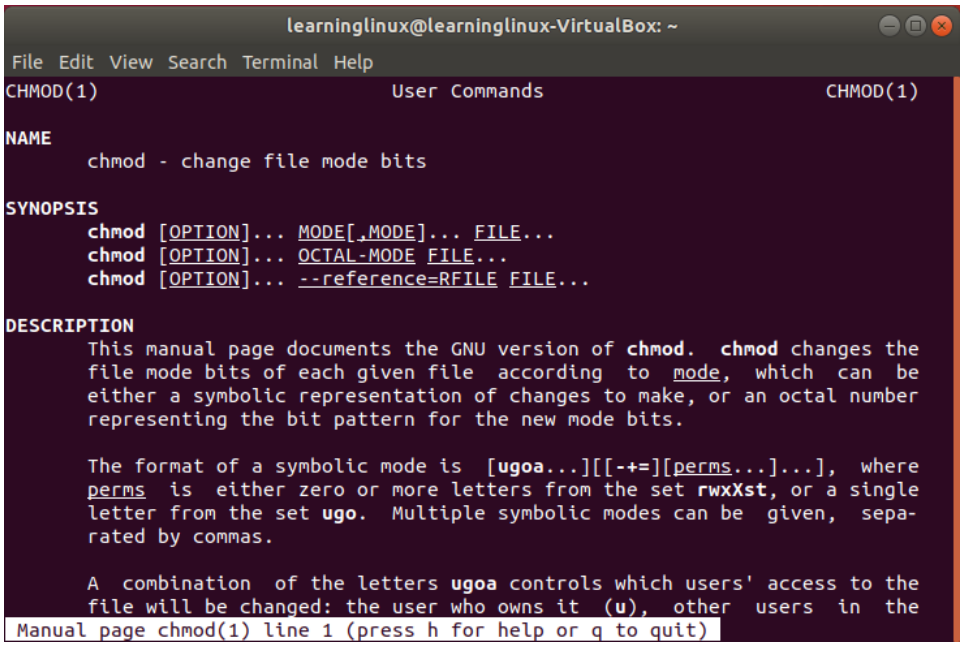
Command	Purpose
ls -l [FILENAME]	list FILENAME information in long format
chmod [MODE] [FILE]	change permissions to MODE for FILE
chmod [OCTAL MODE] [FILE]	change permission to OCTAL MODE for FILE

Chapter 6

Manual Pages

Manual pages are extremely helpful to learn the syntax and arguments for a command. Often, if you cannot run a command properly, the solution can be found on the manual page.

To display all of the optional arguments for a command and to see more information about that command, use the **man** command, which is short for manual. The syntax for the command is `man [COMMAND]`, where the command is what you want to see the manual page of. Go to your terminal and type in *man chmod*.



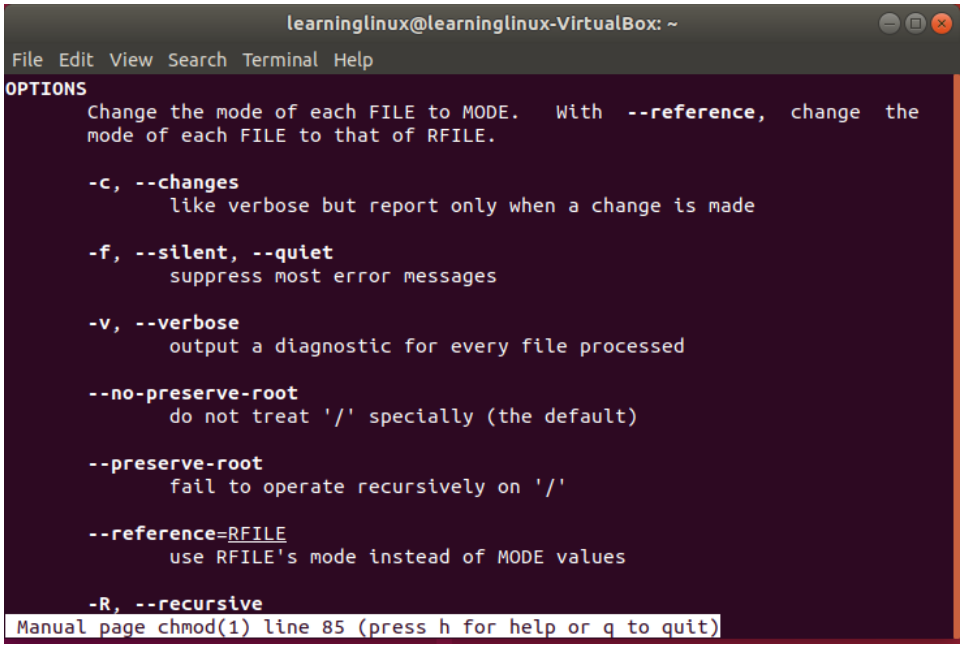
The below are parts of the manual pages for the **chmod** command. All man pages follow the following common format:

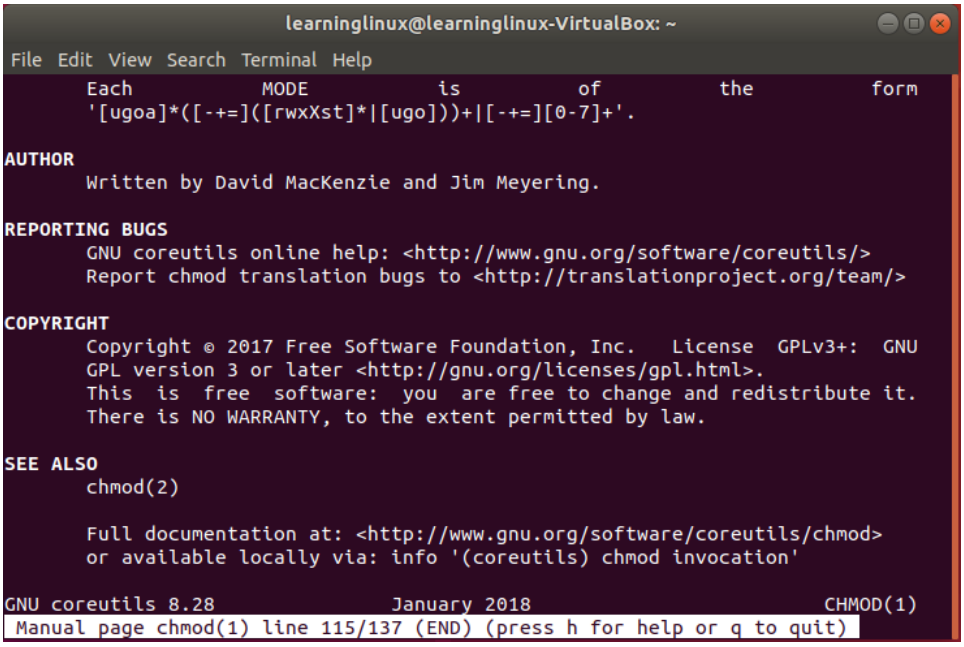
1. **Name** – the name of the command and a short description
2. **Synopsis** – summary of the command with its syntax

3. **Description** – description of the command

4. **Options** – list of optional arguments for the command and their descriptions

5. **See Also** – related commands and link to full documentation online



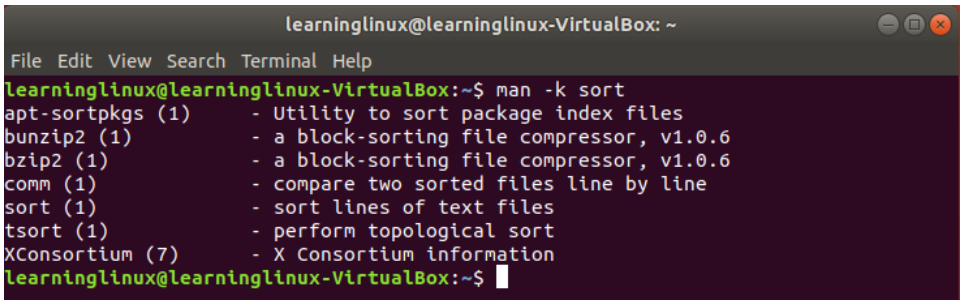


Once you are done with the man page, press “q” to get back to the terminal.

Searching the Manual Pages

If you are unsure of which command to use, you can perform a general search on all of the manual pages to see which ones contain a particular keyword. To do this, use the **man** command with the optional argument `k`, followed by the keyword that you are searching for.

Go to the terminal and type in *man -k sort*.



This search returns a list of all the commands that mention the term "sort". Searching the man pages by keyword is often underutilized, but it is extremely helpful to find the desired command.

Command Summary

Command	Purpose
man [COMMAND]	displays the manual page for COMMAND
man -k [KEYWORD]	searchs all the manual pages for the KEYWORD

Chapter 7

Complex Inputs

Definitions

1. **Complex Input** - input containing characters other than the standard English alphabet and numbers
2. **Wildcard** - characters that are used to substitute unknown characters in a search
3. **Escape Character** - a character that changes the interpretation of the subsequent character

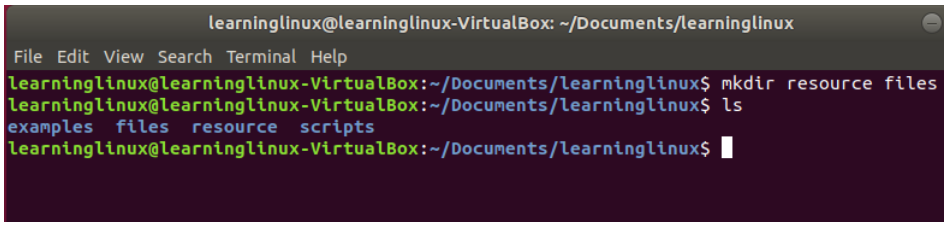
Complex Input

So far all of the commands that have been introduced use input that contains only alphanumeric characters. However, commands also use and interpret complex inputs, which contain wildcards, escape characters, whitespace, or text enclosed in quotes. Examples of complex inputs are as follows:

1. *.txt
2. myfile.???
3. “Hello World”
4. \n
5. \t
6. ’$PATH’

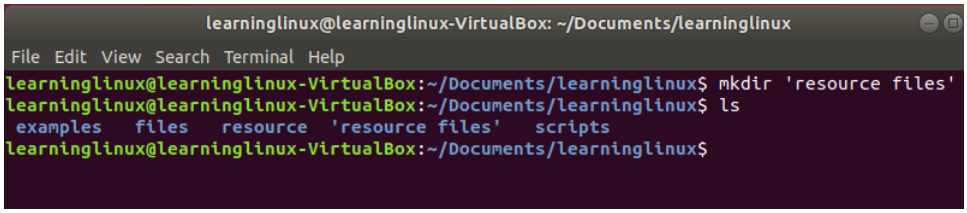
Spaces (Whitespace)

Spaces are often the most overlooked complex input. In the shell, whitespace indicates the end of an argument or command and the shell splits the commands and arguments accordingly. A common issue that can occur is moving into a directory whose name contains a space.

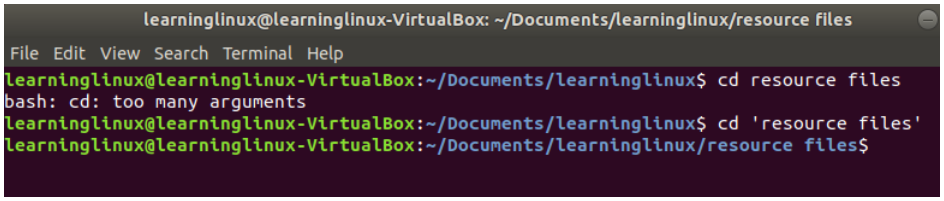


For example, you wish to create a single folder named `resource files`. If you use the command *mkdir resource files*, the shell interprets two separate names and creates two new directories.

The proper way to make a directory (or file) that contains a space is to put the name in quotes. Go to the terminal and type *mkdir 'resource files'*.



Another issue that occurs with directory (or file) names that contain spaces is with the **cd** command. Type *cd resource files* in the terminal, and it displays an error. To avoid the problem, enclose the directory name in quotes: *cd 'resource files'*.



Note: Avoid using spaces in filenames or directory names. This makes it difficult to perform operations on such files and directories.

Quoting (Strong & Weak)

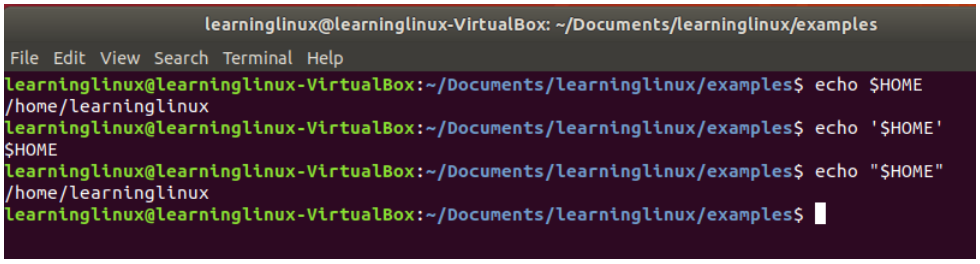
Quoting is a method of encapsulation that allows the shell to interpret commands and arguments that may have spaces, other complex inputs, or special characters reserved for the shell. There are two types of quoting:

1. **Strong Quoting** - uses the single quote, and interprets everything literally (as it is)
2. **Weak Quoting** - uses the double quote, and the shell will evaluate variables

To see the differences between strong and weak quoting, use the **echo** command. The syntax for this command is `echo [STRING]`. The command will display the string to the standard output. Go to the terminal and type *echo $HOME*. The `$HOME` is a special variable in the shell that is reserved for the home directory. This command will display the home directory on the command line.

Now try the command *echo '$HOME'*. This time, $HOME is enclosed by single quotes and the result is different. The output is the string '$HOME'. Since single quotes are used here, the shell interprets the string literally and does not expand it.

Lastly, try the command *echo "$HOME"*. This displays the home directory as in the first example. Since double quotes are used, the shell will expand the variable and output the expansion to the command line.



Wildcards

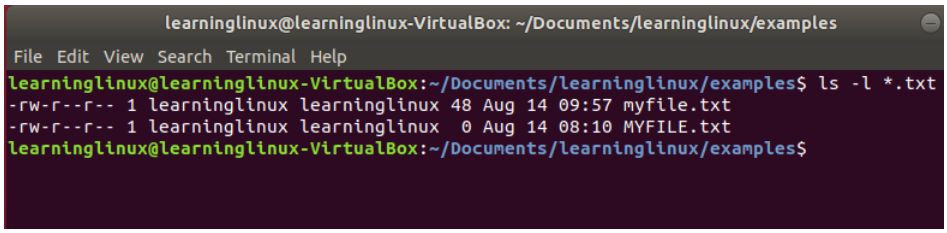
Wildcards are characters used as substitutes for one or more characters in a search, giving the search more flexibility. Three types of wildcards are used in a Linux environment:

1. * (star wildcard)
2. ? (question mark wildcard)
3. [] (square brackets wildcard)

When a command uses a wildcard, the shell will interpret the wildcard, perform the query, and then use the results to execute the rest of the command.

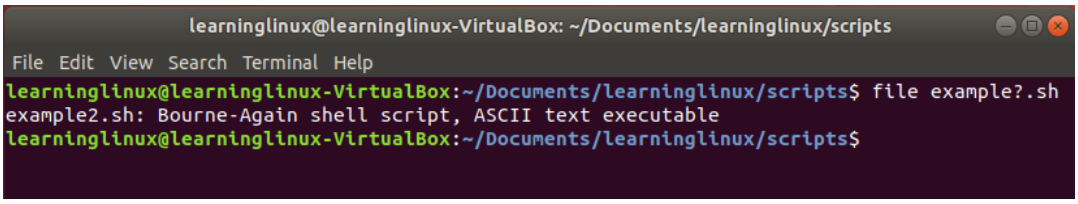
Star Wildcard

The star wildcard is the asterisk. This is the most versatile wildcard. It can represent any number of characters. Navigate to a directory that contains text files and run the command *ls -l *.txt*. This command will search for any file or directory in the directory with an ending of `.txt`. The shell will transform this to *ls -l myfile.txt MYFILE.txt*.



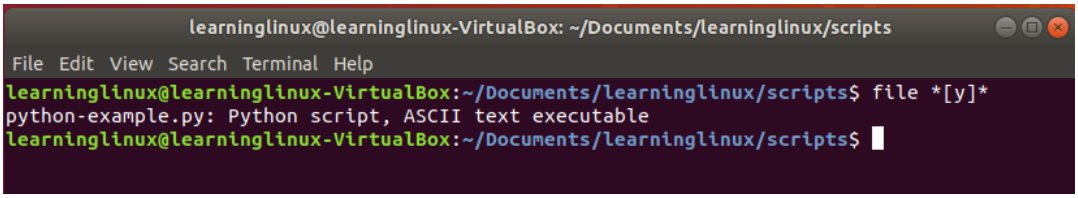
Question Mark Wildcard

The next wildcard is the question mark (?). This wildcard represents only one character. Navigate to the `scripts` directory and type in the command *file example?.sh*. The shell transforms this to *file example2.sh*.



Square Brackets Wildcard

The last wildcard are the square brackets ([]). This wildcard substitutes all the characters that are enclosed within the brackets. In the same `scripts` directory, type in the command *file *[y]**. This command will search for anything in the directory that contains the character `y`. The shell transforms the command to *file python-example.py*.



Escape Characters

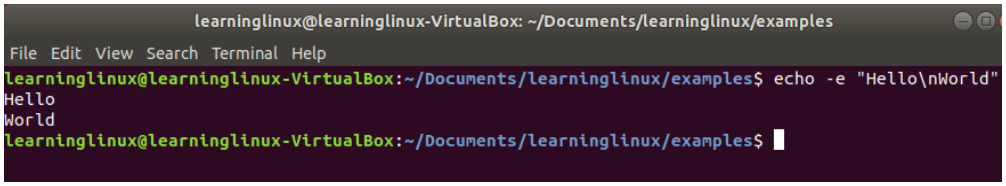
Escape characters are important when using the command line because they change the meaning of the next sequence of characters so that the shell interprets them in a particular manner.

The following are commonly used escape characters.

Escape Character	Interpretation
\”	double-quote
\’	single-quote
\n	newline
\t	horizontal tab
\b	backspace

Earlier there was an issue with spaces in the name of a directory. Recall the command: *mkdir ’resource files’*. The name of the directory was enclosed in single quotes so that the shell could interpret the name of the directory properly. However, an alternative command could be used: *mkdir resource files*. In this command, the escape character is ‘‘ ’’, which is a backslash followed by a space. When this command is processed, the shell will interpret the escape character as a space and therefore, the directory name does not need to be enclosed in quotes.

A simple example of an escape character used in a command can be shown by using the echo command again. Try the command *echo -e “Hello\nWorld”*. The `e` argument allows the echo command to interpret escape characters.



Command Summary

Command	Purpose
echo [STRING]	displays the STRING to the standard output

Chapter 8

Process Management

Definitions

1. **Process** - a program that is being executed
2. **Foreground Process** - process that is active in the terminal and may require additional input
3. **Background Process** - process that runs without being connected to an input source

Foreground and Background Processes

There are two ways to run a process, either in the foreground or in the background. A foreground process takes input from the keyboard and outputs to the screen or terminal. All of the commands so far have been processes that run in the foreground.

A background process operates without being connected to the keyboard. If a background process requires input, it will wait for the input. If a process runs in the background, then other commands and processes can be started from the terminal and they do not need to wait for the background process to complete. To have a process run in the background, put an ampersand (`&`) symbol at the end of a command. An example of this is provided later in this section.

Process States

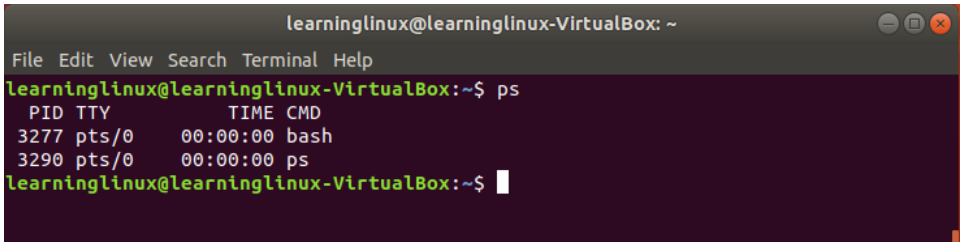
A process state changes over time as the Linux environment changes. There are four process states in Linux:

1. **Running** - current process in the system or waiting to be assigned to a CPU
2. **Waiting** - process is waiting for system resources or for an event to occur

3. **Stopped** - process is not running
4. **Zombie** - process has been completely halted, but still has an entry in the process table

Viewing Active Process

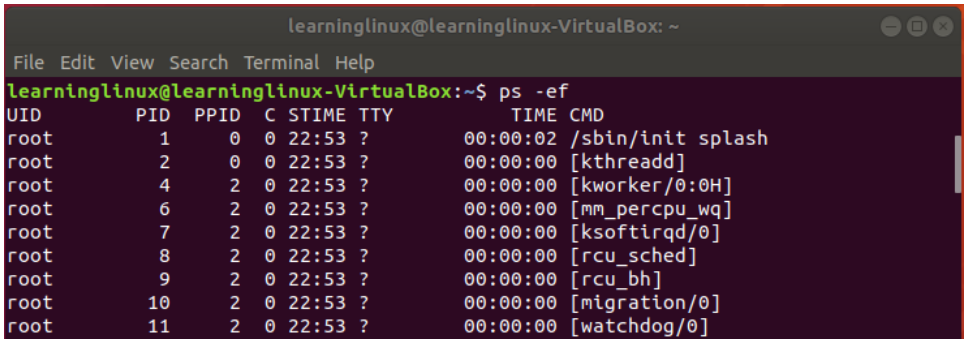
Two commands can be used to view the active processes on the machine: **ps** and **top**. **ps** stands for process status. The syntax for this command is `ps [OPTIONS]`. There are many optional arguments for this command and they can be used in various combinations. Go to the terminal and type *ps*.



The output displays four columns of information:

1. **PID** - the process ID; each process is assigned a unique number
2. **TTY** - terminal type associated with the process
3. **TIME** - CPU time used by the process
4. **CMD** - the command that started the process

Some additional arguments that can be added to the **ps** command are `e` and `f`. The `e` argument will list all the processes in the system and the `f` argument shows all the information about the process. Try the command *ps -ef* in the terminal.

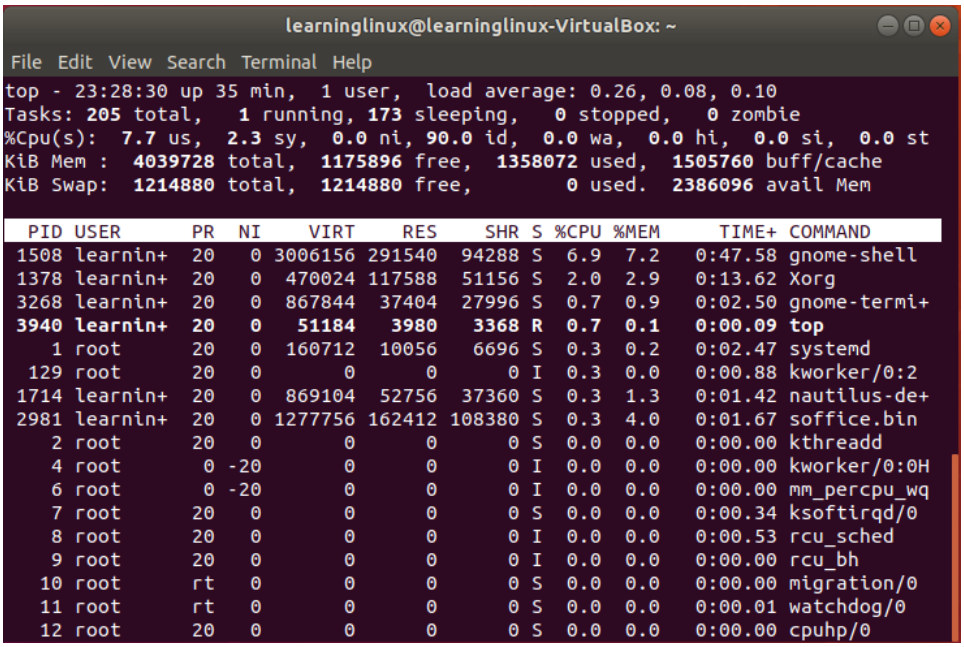


There is more output than is displayed in the picture above. There are now eight columns of information for each process:

1. **UID** – user ID of the person who ran the process

2. **PID** – [SAME AS ABOVE]
3. **PPID** – parent process ID, the ID of the process that started this process
4. **C** – CPU utilization of the process
5. **STIME** – time that the process was started
6. **TTY** – [SAME AS ABOVE]
7. **TIME** – [SAME AS ABOVE]
8. **CMD** – [SAME AS ABOVE]

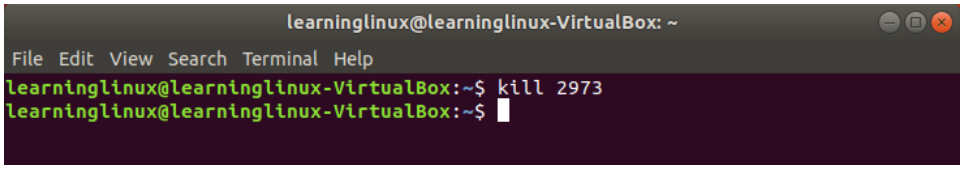
The other command to view active processes is **top**. The syntax for the command is `top [OPTIONS]`. Go to the terminal and type *top*.



Killing Processes

If a process has frozen and becomes unresponsive, you may have to end it. To end a process, use the **kill** command. The syntax for this command is `kill [OPTIONS] [PID]`. To kill a process, the process id (pid) must be known, and that is how the operating system knows which process to end.

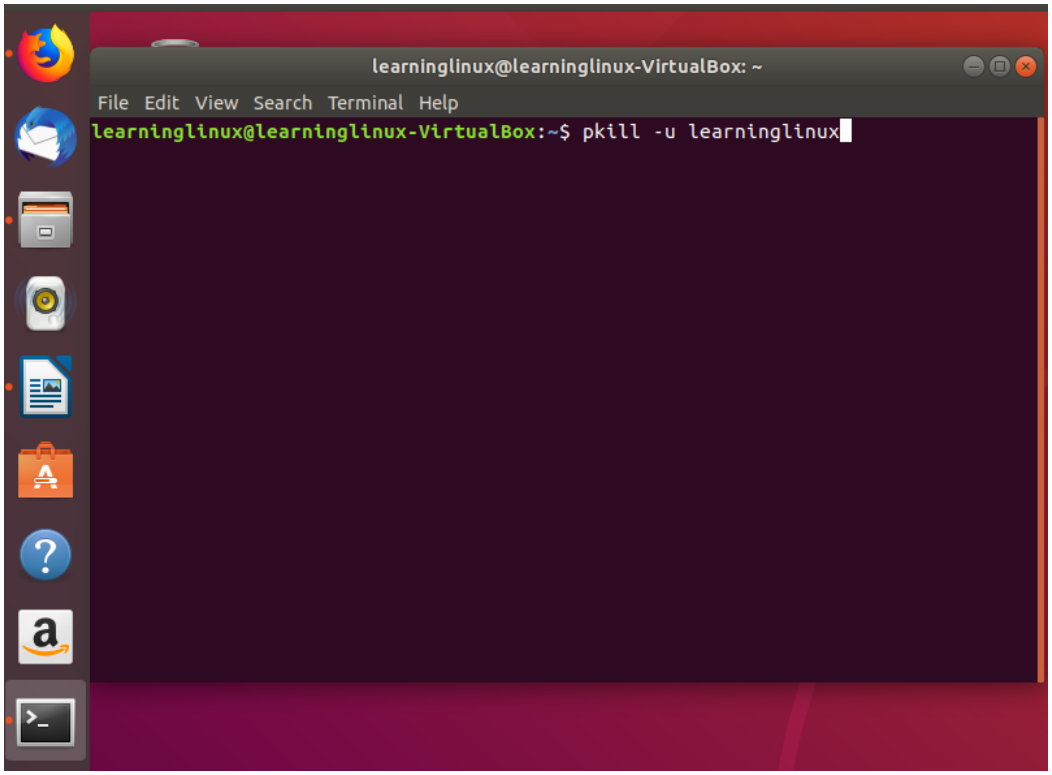
For example, say Firefox was opened, but became unresponsive and had to be killed through the command line. First, run the **ps** command to locate the pid number for the Firefox process. The next section will explain how to find the pid quickly without having to read all the output of **ps**. Upon a quick search, the pid is 2973. Then type in the terminal *kill 2973*.



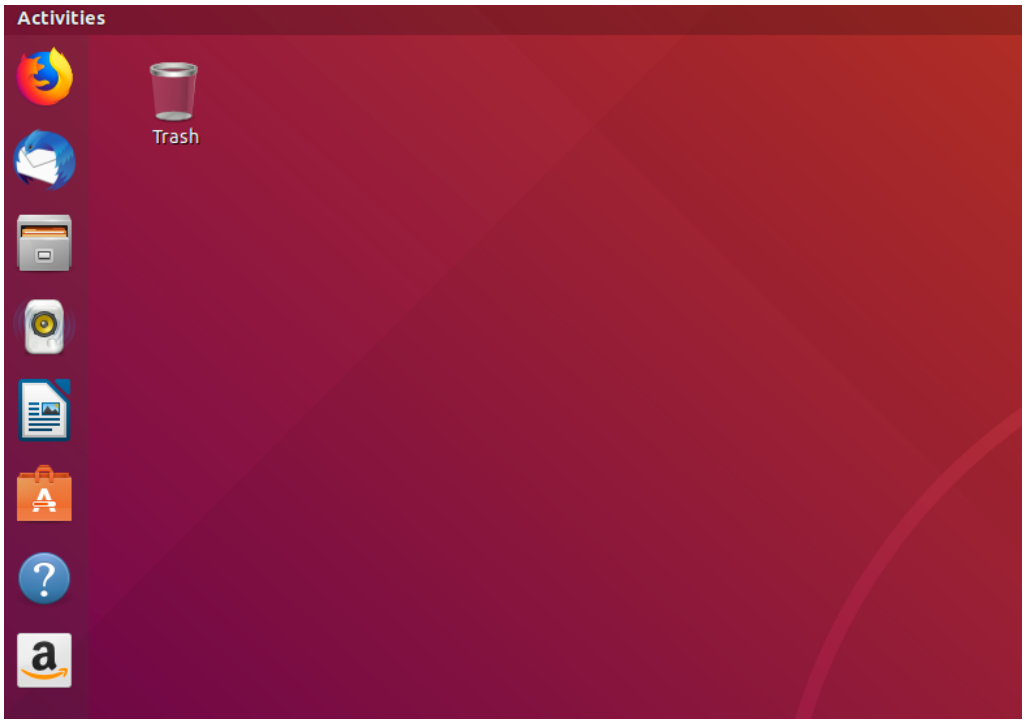
If the ps command is run again, no process should be listed relating to Firefox.

Now if there are multiple processes that are frozen, unresponsive, or that are replicating at an exponential rate, there is a command to end all processes. That command is `pkill`. The syntax for this command is `pkill -u [USERNAME]`. Applying the optional `u` argument along with the username will kill **ALL** processes that were started by the user.

A lot of my processes such as Firefox, word processor, and the file explorer are now unresponsive and I want to end all of them. To do so, I would type in my terminal: *pkill -u learninglinux*.



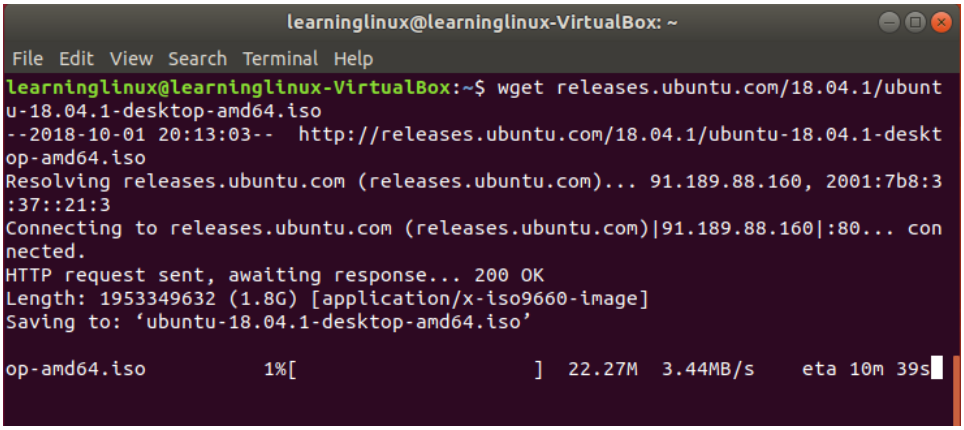
After executing this command, none of the programs are running, and the terminal was stopped as well.



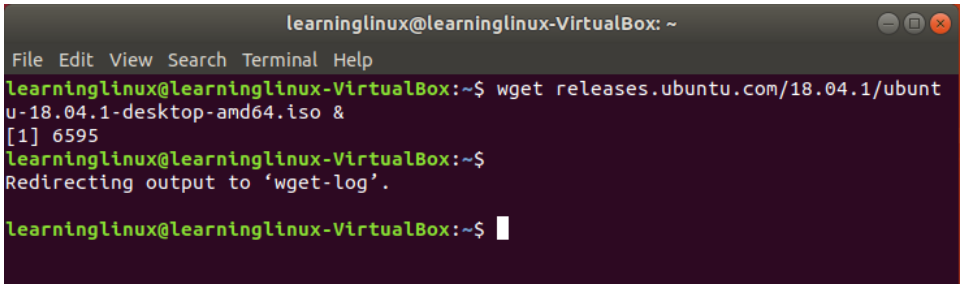
Moving Processes To and From the Background and Foreground

Running processes in the background can be convenient because you can still use the command line while the process runs. For example, downloading a file from the command line will display the progress of the download, but this may be time consuming. The command to download a file is **wget**. The syntax for this command is `wget [URL]`. The command *wget releases.ubuntu.com/18.04/1/ubuntu-18.04.1-desktop-amd64.iso* will download the operating system file for Ubuntu 18.04 LTS. This is a large file and this download would take at least 10 min.

Warning: Do not try this command in the terminal

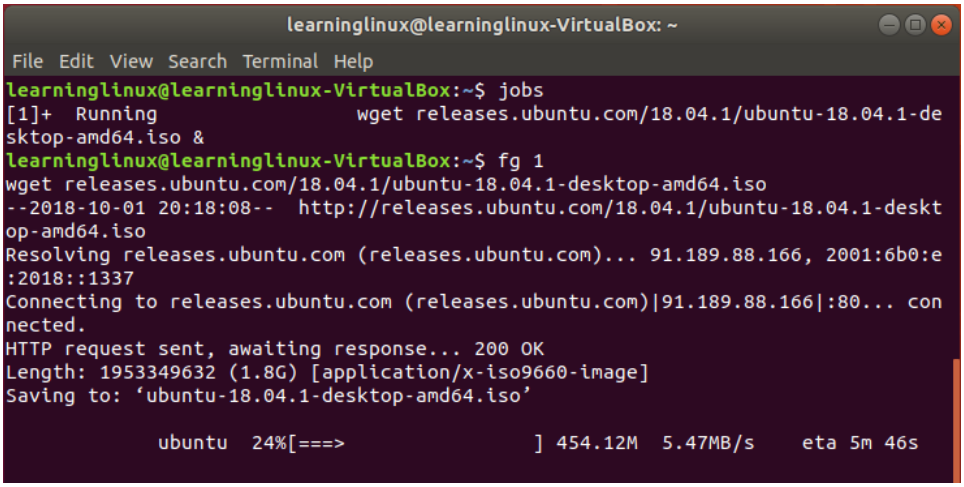


During the download, the terminal cannot be used since it displays the progress of the download in the foreground. However, by moving the process to the background, the command line can be used to do other things as the file is being downloaded. To have a process run in the background, add the ampersand (&) symbol directly after the command. Now the command would be *wget releases.ubuntu.com/18.04/1/ubuntu-18.04.1-desktop-amd64.iso &*.



The process is running in the background now and will continue to run until it is complete or it is stopped by the operating system.

To view the processes that are running in the background, use the **jobs** command. The following is the output of the **jobs** command and the download status is moved to the foreground.



To move the most recnt background process to the foreground, use the **fg** command. The other way to move a process back to the foreground is to specify the job number as an argument. The jobs command displays the **job** number associated with a background process.

Command Summary

Command	Purpose
ps [OPTIONS]	displays all the active processes
top	displays all the active processes
kill [PID]	kill the process with the given PID
pkill -u [USERNAME]	kills all processes started by USERNAME
wget [URL]	downloads a file from the URL
jobs	displays the status of all the current jobs
fg	moves the most recent job to the foreground

Chapter 9

Filters and Regular Expressions

Definitions

1. **Regular Expression** - sequence of characters that define a search pattern

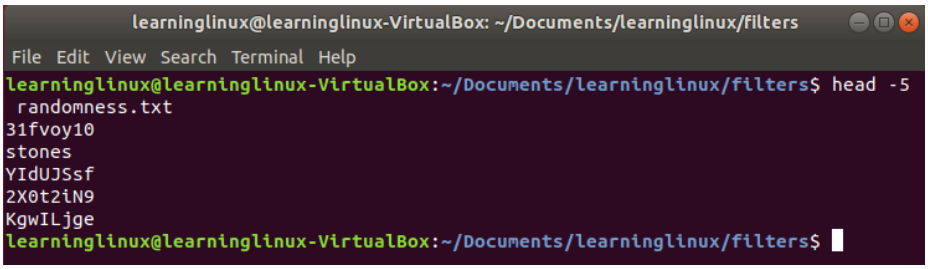
Filtering

Filtering allows you to search through data in specified ways. There are basic filtering techniques such as getting the first five items of a list, or sorting the items alphabetically. Advanced filtering techniques make use of regular expressions (regex) in order to search for very specific data items.

The examples in this section make use of a file called `randomness.txt`, which includes 25 alphanumeric strings, some of which are words.

Head

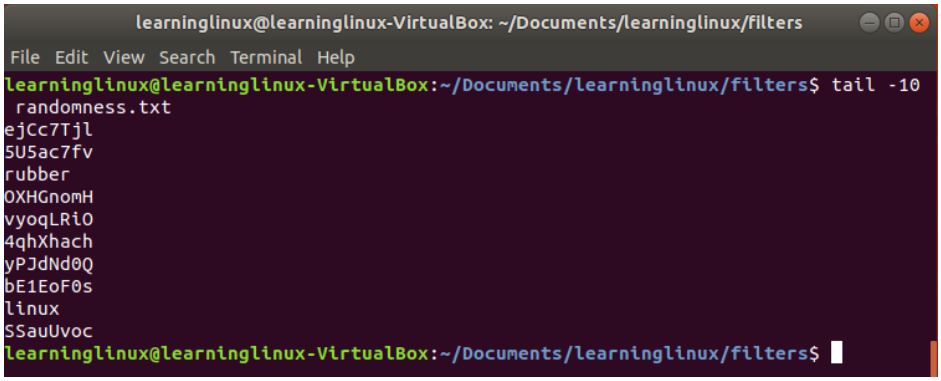
The **head** command outputs the first `n` number of lines. The syntax for this command is `head [OPTIONS] [FILE]`. To see the first 5 lines of the `randomness.txt` file, go to the terminal and type in *head -5 randomness.txt*



The output is the first five lines of the file.

Tail

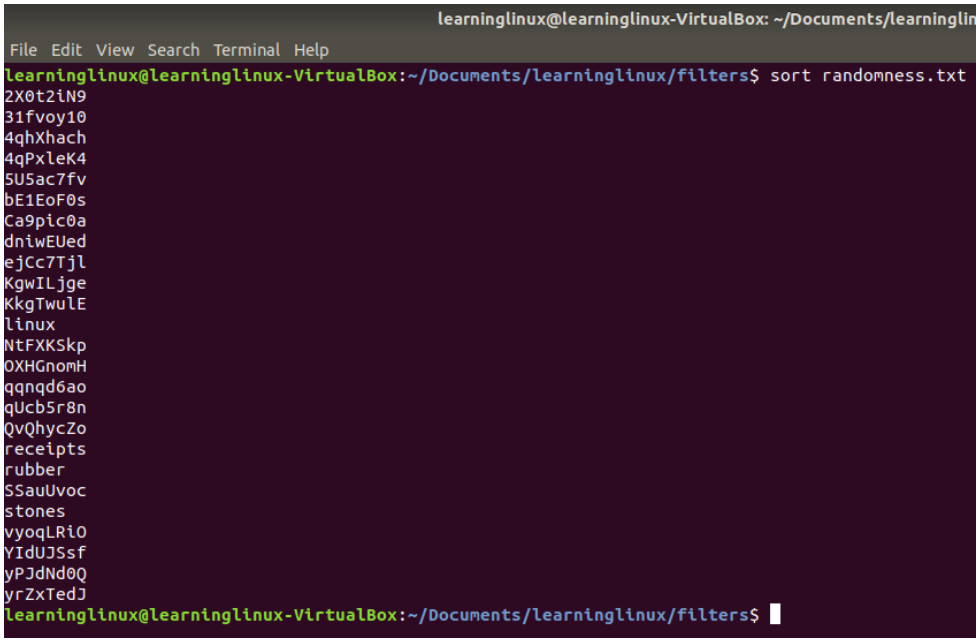
The **tail** command is just like the **head** command, but it will output the last `n` number of lines. The syntax for the **tail** command is `tail [OPTIONS] [FILE]`. To see the last 10 lines of the `randomness.txt` file, go to the terminal and type in *tail -10 randomness.txt*.



The output is the last 10 lines of the `randomness.txt` file.

Sorting

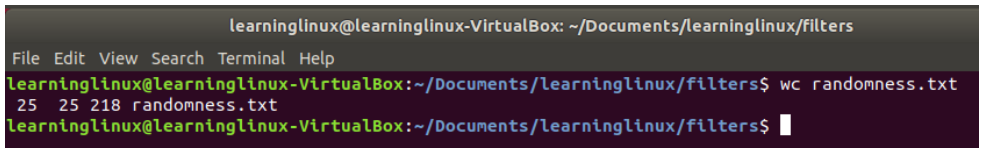
Sorting is a powerful filtering technique used in many algorithms and applications. To sort the contents of a file, use the **sort** command. The syntax is `sort [OPTIONS] [FILE]`. To sort the `randomness.txt` file, go to the terminal and type in *sort randomness.txt*.



By default the sort command will sort in alphabetical order, but additional arguments can be specified in the command to sort by different categories.

Word Count

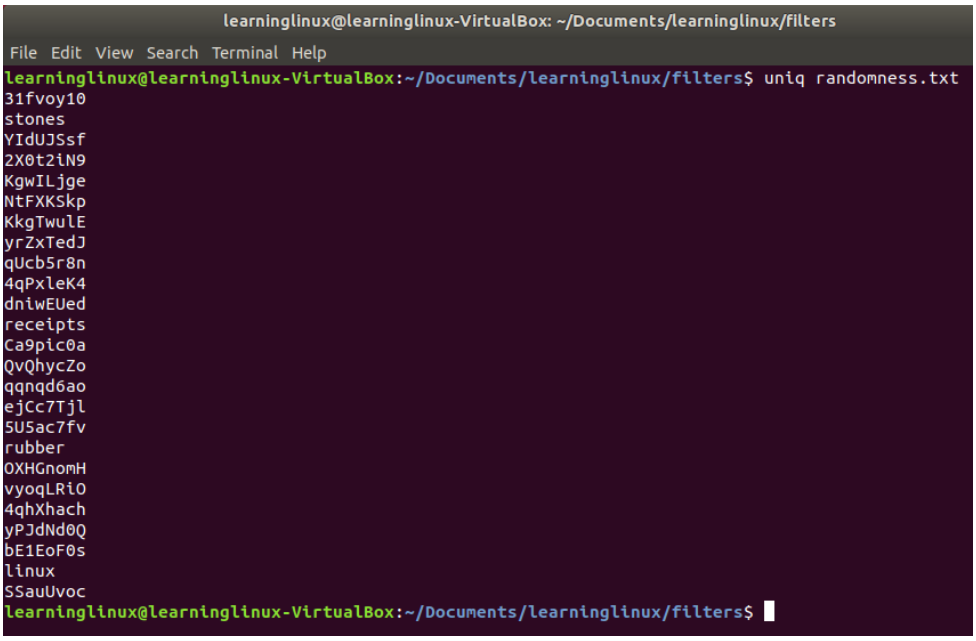
Another filtering technique is to get the word count of a file. To do this, use the **wc** command, which stands for word count. The syntax for the **wc** command is `wc [OPTIONS] [FILE]`. To get the word count of the `randomness.txt` file, go to the terminal and type in *wc randomness.txt*.



The output displays the number of words, which is 25; it also displays the number of characters in the file as well.

Unique Items

In some cases, there may be duplicate items in a data set. To search for the unique items in a data set, use the **uniq** command, which stands for unique. The syntax for this command is `uniq [OPTIONS] [INPUT]`. Go to the terminal and type in *uniq randomness.txt*.



The output is all the strings in the file minus any repeated lines. However, it can be difficult to determine if any lines have been omitted or not. This can be easily done with piping which is discussed in the next chapter.

Regular Expressions

A regular expression, also called regex, is a combination of wildcards and other information which builds a particular search pattern. Regex is a powerful filtering tool, but it is often difficult to understand. Only a few basic examples are covered in this section.

Regex Syntax

The following are the various characters that are used in a regex:

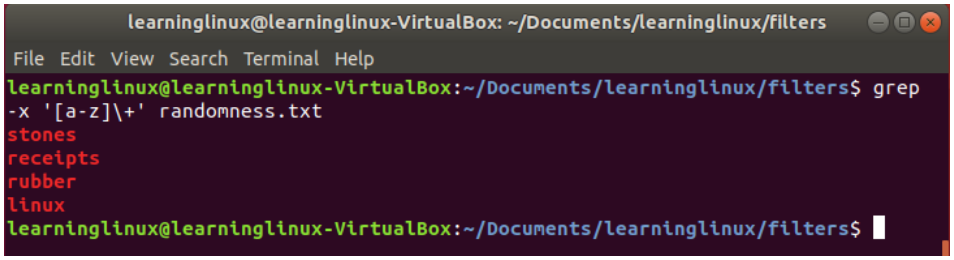
1. . (dot) – a single character
2. ? - preceding character matches zero or one times
3. * - preceding character matches zero or more times
4. + - preceding character matches one or more times
5. [a-z] – character is included in the range of characters specified by the brackets
6. [^ a-z] – character is not one of those included in the brackets in that range

7. {n} – preceding character matches exactly n times
8. {n, m} – preceding character matches at least n times, but not more than m times
9. ^ – matches at the beginning of the line
10. $ – matches at the end of the line

The above can be combined in various ways in order to build the desired pattern.

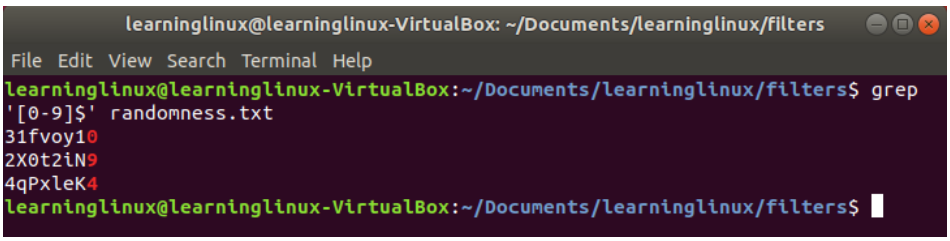
Several different commands can use regex in order to find a matching pattern. For all the following examples, we use the **grep** command, which stands for global regular expression print. The syntax of this command is `grep [OPTIONS] PATTERN [FILE]`.

For the first example, we try to find words that only contain lowercase letters. The regex for this would be `'[a-z]\+'`. [a-z] covers all the lowercase letters in the alphabet and the + will match at least one or more lowercase letter. Now, the grep command would be *grep -x '[a-z]\+' randomness.txt*. Pay attention to the `x` optional argument. This will search entire lines instead of individual characters.



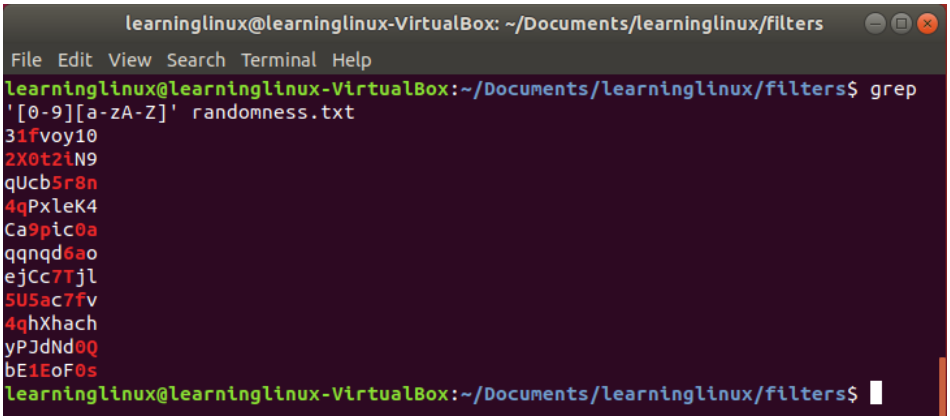
As seen in the output above, only the words with all lowercase letters are displayed.

For the next example, we try to search for lines that end with a number. The regex for this would be `'[0-9]$'`. The [0-9] represents all the numbers, and the '$' matches the end of the line for the numbers. Now, the **grep** command would be *grep '[0-9]$' randomness.txt*.



As seen in the output above, there are three lines that end in a number.

For the last example, we will try to find the lines that have a number followed by a letter. The regex for this would be `'[0-9][a-zA-Z]'`. The [0-9] represents the numbers and the [a-zA-Z] represents all upper and lowercase letters. The grep command would be *grep '[0-9][a-zA-Z]' randomness.txt*.



Command Summary

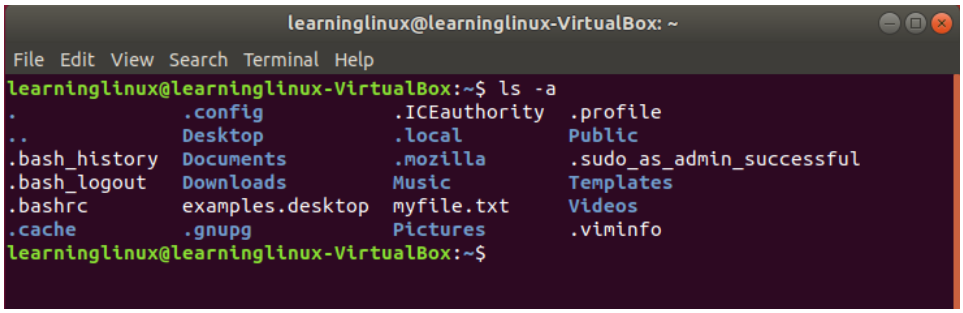
Command	Purpose
head [OPTIONS] [FILE]	return lines from the beginning of the FILE
tail [OPTIONS] [FILE]	return lines from the end of the FILE
sort [OPTIONS] [FILE]	sort the lines of FILE alphabetically
uniq [OPTIONS] [INPUT]	return non-repeating lines from the INPUT
wc [OPTIONS] [FILE]	give the word count of FILE
grep [OPTIONS] PATTERN [FILE]	search for the PATTERN in the lines of FILE

Chapter 10

Input Output Redirection

Input & Output

Many of the examples used throughout the sections take advantage of inputting a command and awaiting for a particular output. These input and output are are known as data streams, which can take many forms.



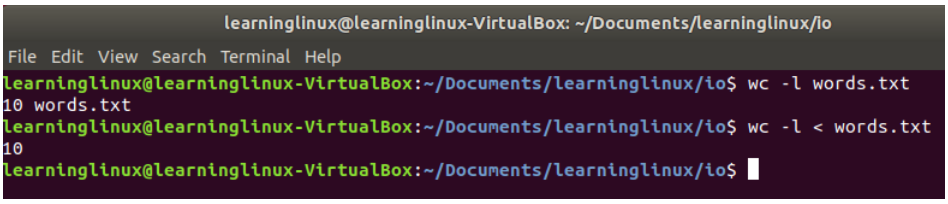
In this example, the input is the command *ls -a*, which is taken from the keyboard. And the output displays a list of all the files and directories in the terminal.

These data streams can be put into three categories:

1. **Standard Input** (stdin): input fed into the program, defaults to the keyboard
2. **Standard Output** (stdout): output of the program, defaults to the terminal
3. **Standard Error** (stderr): error messages of the program, defaults to the terminal

Redirecting Input

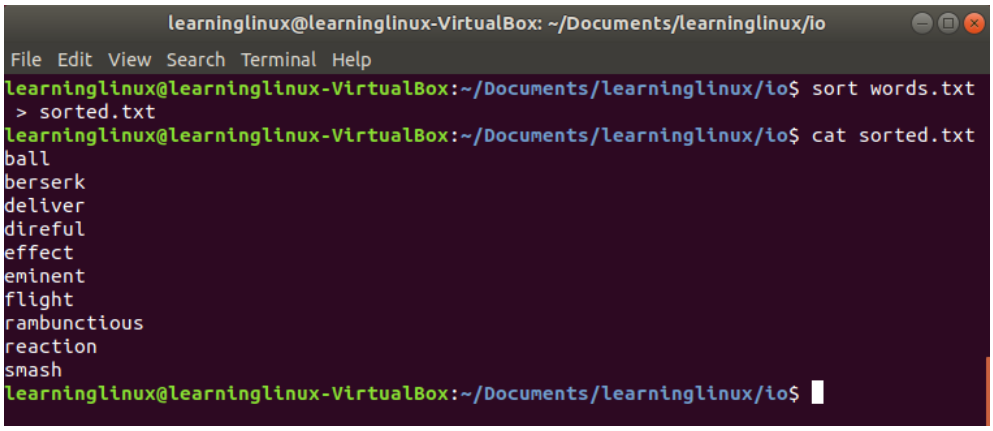
Commands can take input from a source other than the keyboard by using the "<" symbol. To determine the number of lines in a file, use the **wc** command. The syntax for this command is `wc [FILE] [OPTIONS]`. Type in *wc -l words.txt* in the terminal. This will output the number of lines in the file and the filename.



Now, type in *wc -l < words.txt*. The output is just 10. That is because the wc command uses input redirect from the file, and therefore does not know the filename.

Redirecting Output

Commands can redirect their output to places other than the terminal. For example, instead of outputting the results directly to the terminal, they can be put in a file. To redirect the output, use the ">" and then specify where the output should go. To sort ten words and put all the sorted words in a file, use the **sort** command. Type in the command *sort words.txt > sorted.txt*. This will sort all the words in the words.txt file and put the results in the sorted.txt file.

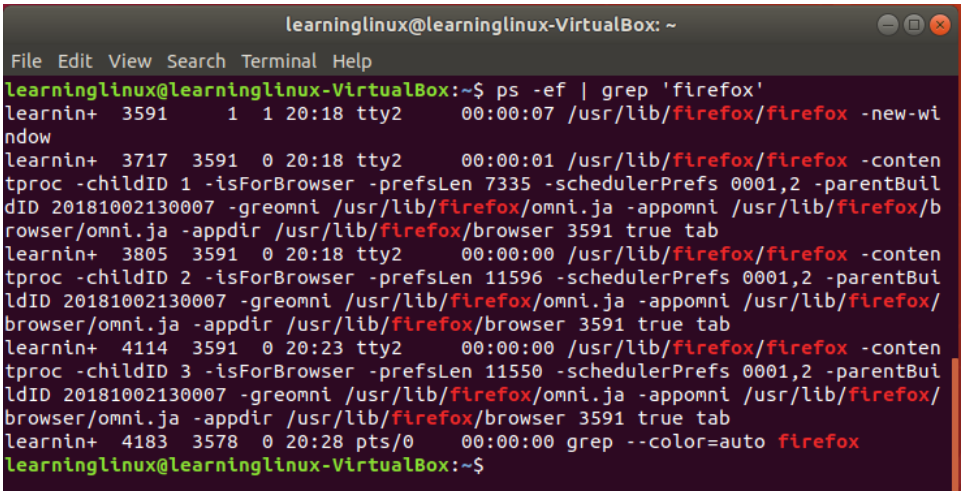


Then use the **cat** command to view the sorted words. Type in *cat sorted.txt*.

Piping and Command Chaining

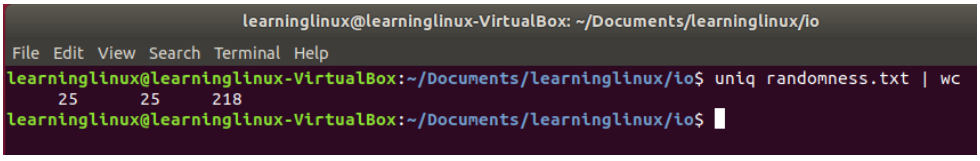
Piping redirects the output of one command to the input of another command. Piping is very powerful because it enables you to use several commands without having to save the output to temporary files and such.

In the process management chapter, we had to parse through the output of the command *ps -ef* in order to find the process ID of a certain program that was running in order to kill the program. Using pipes streamlines this process even further. Try the command *ps -ef | grep 'firefox'*.



The output of this command is a trimmed version of the output of the *ps -ef* command. As seen in the before, the **grep** command will return all the lines that contain the given word. And combining these two commands finding the proper process ID simpler.

In the filtering and regular expressions chapter, we ran the **uniq** command to display the unique items in the `randomness.txt` file. However, it was difficult to determine if any repeated items were removed. To determine if any items were repeated in the `randomness.txt` file, we can combine the output of the **uniq** command with the **wc** command. Go to the terminal and type in *uniq randomness.txt | wc*.



The output of this chain of commands is 25. And since the original file has 25 strings in it, we know that there were no repeated items.

Chapter 11

Basic Networking

Definitions

1. **Hostname** - label assigned to a device connected to a computer network

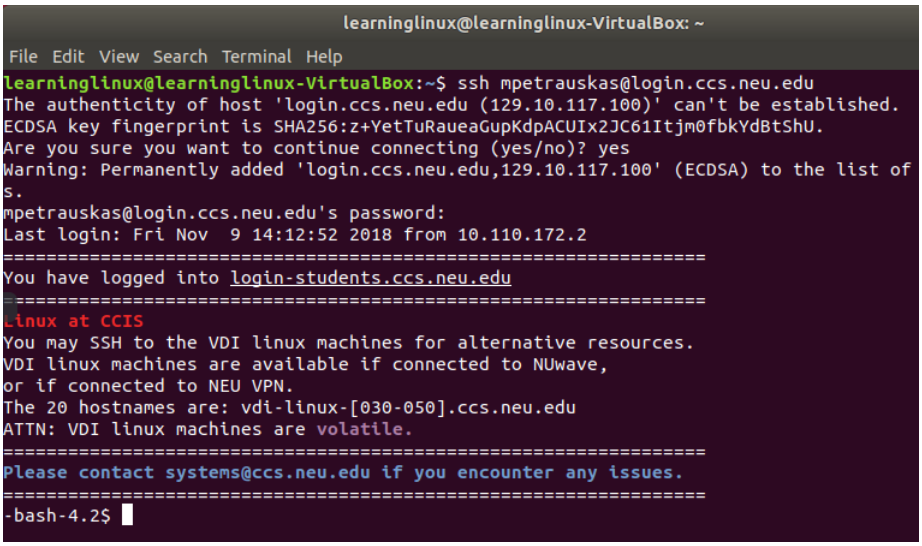
Networking

Networking enables you to connect to, send, and receive information from other machines.

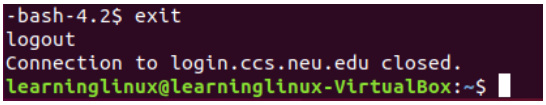
Secure Shell

The most important method of communicating with another computer with the shell is via *secure shell* or **ssh**. The **ssh** command will run the OpenSSH SSH client, which is a remote login program. The syntax for the **ssh** command is `ssh [OPTIONS] [HOSTNAME]`. In general, only the hostname needs to be provided with the **ssh** command, but there are cases where a username is needed. One such example is when you need to login to the CCIS Servers. To login to these servers, you have to use the following format: `ssh username@hostname`. More specifically, for the CCIS Servers, it would be `ssh [your username]@login.ccs.neu.edu`.

Go to the terminal and try to log in to the CCIS Server.



You will be prompted for a password and once you are connected to the CCIS Server you will be in an entirely new bash shell that is run by the server.

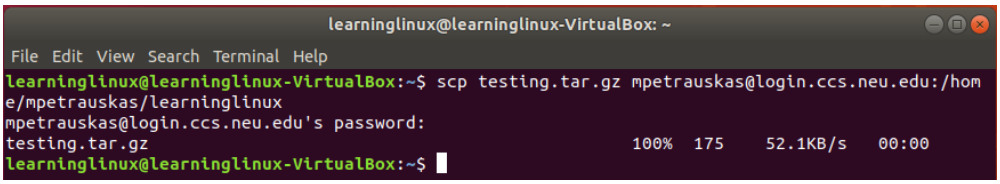


Once you are done on the remote server, you can use the **exit** command to close the connection to the CCIS server and return to the shell on your localhost.

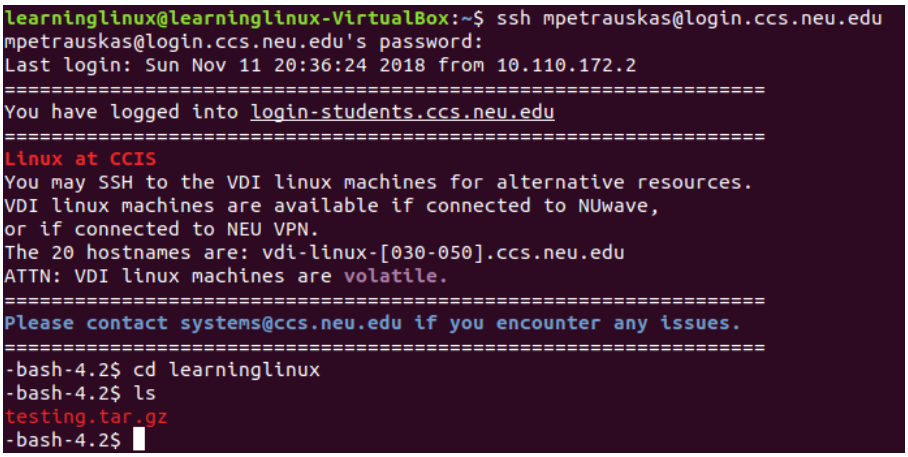
Secure Copy

Simply connecting to another machine may not be enough. Often, files need to be transferred between machines. In order to do this, use the **scp command**, which stands for secure copy. The syntax for the **scp** command is `scp [OPTIONS] [LOCALHOST] [REMOTEHOST]`. This command can be used to transfer files from your localhost (your machine) to a remote host (the CCIS Server) or vice versa. It can also be used to transfer files from two remote hosts (e.g. from the CCIS server to another external server).

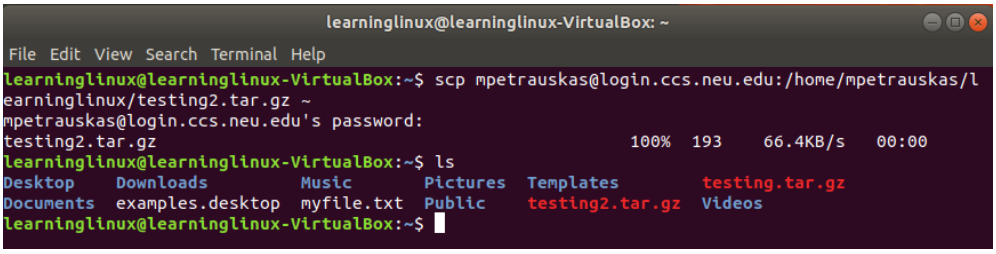
To transfer a file from the localhost to a remote host, the **scp** command has the following syntax: `scp [FILENAME] [REMOTEHOST FILEPATH]`. To transfer a file, I would type in *scp testing.tar.gz mpetrauskas@login.ccs.neu.edu:/home/mpetrauskas/learninglinux*



The file was successfully transferred to the CCIS Server, and we can see the file once we log in into the server.



To transfer a file from a remote host to a localhost, the **scp** command has the following syntax: `scp [REMOTEHOST FILEPATH] [LOCALHOST FILEPATH]`. The file will be transferred from the remote host at the given filepath to the filepath on the localhost. To transfer a file, I would type in *scp mpetrauskas@login.ccs.neu.edu:/home/learninglinux/testing2.tar.gz* ~. Note that the "~" at the end of the command indicates the home directory on the virtual machine.



It is simple to verify that the file has been successfully transferred back to the localhost. Check your home directory for the file. Now, try this out on some files and move them between your virtual machine and the CCIS Server.

Note: the **scp** command can also transfer entire directories between machines.

Command Summary

Command	Purpose
ssh [HOSTNAME]	remotely connect to the specified HOSTNAME
scp [LOCALHOST] [REMOTEHOST]	copies the file at the LOCALHOST filepath to the REMOTEHOST filepath
scp [REMOTEHOST] [LOCALHOST]	copies the file at the REMOTEHOST filepath to the LOCALHOST filepath
exit	closes the active bash shell

Chapter 12

Command Cheat Sheet

The following is a table of the most important commands covered in this guide and should be used as a reference when the syntax or purpose of a command is needed.

File System Navigation

Command	Purpose
pwd	print name of current/working directory
cd [DIR]	change the working directory to DIR
ls	list directory contents

File Manipulation

Command	Purpose
file [FILENAME]	Determines the file type of FILENAME
mkdir [DIRECTORY]	Make the directory DIRECTORY
cp [SOURCE] [DEST]	Copy the SOURCE TO DEST
rm [FILENAME]	Remove or delete FILENAME
mv [SOURCE] [DIRECTORY]	Move the SOURCE file or directory to DIRECTORY
mv [SOURCE] [DEST]	Rename the SOURCE file or directory to DEST

Process Management

Command	Purpose
ps [OPTIONS]	displays all the active processes
top	displays all the active processes
kill [PID]	kill the process with the given PID
wget [URL]	downloads a file from the URL
jobs	displays the status of all the current jobs
fg	moves the most recent job to the foreground

Filtering

Command	Purpose
head [OPTIONS] [FILE]	return lines from the beginning of the FILE
tail [OPTIONS] [FILE]	return lines from the end of the FILE
sort [OPTIONS] [FILE]	sort the lines of FILE alphabetically
uniq [OPTIONS] [INPUT]	return non-repeating lines from the INPUT
wc [OPTIONS] [FILE]	give the word count of FILE
grep [OPTIONS] PATTERN [FILE]	search for the PATTERN in the lines of FILE

Networking

Command	Purpose
ssh [HOSTNAME]	remotely connect to the specified HOSTNAME
scp [LOCALHOST] [REMOTEHOST]	copies the file at the LOCALHOST filepath to the REMOTEHOST filepath
scp [REMOTEHOST] [LOCALHOST]	copies the file at the REMOTEHOST filepath to the LOCALHOST filepath
exit	closes the active bash shell

Miscellaneous

Command	Purpose
chmod [MODE] [FILE]	change permissions to MODE for FILE
chmod [OCTAL MODE] [FILE]	change permission to OCTAL MODE for FILE
man [COMMAND]	displays the manual page for COMMAND
man -k [KEYWORD]	searchs all the manual pages for the KEYWORD
echo [STRING]	displays the STRING to the standard output