IS 340 Operating Systems
HOP01 – Your First Virtual Machine in Virtualbox

3/22/2021 Developed by Matt Raio
3/31/2021 Reviewd by Pengfei(Chuck) Liu
02/25/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

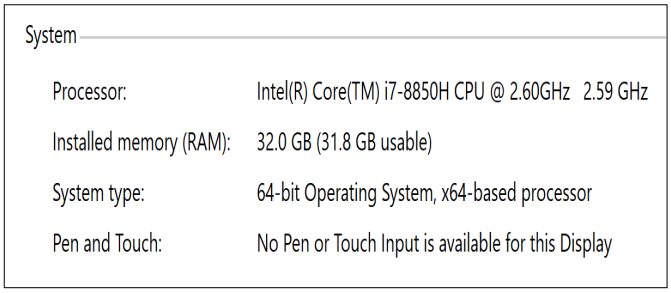
Before You Start

- Version numbers may not match with the most current version at the time of writing. If given the option to choose between stable release (long-term support) or most recent, please choose the stable release rather than beta-testing version.
- This tutorial targets new users with limited experience with Linux.
- There might be subtle discrepancies along the steps. Please use your best judgment while going through this cookbook style tutorial to complete each step.
- For your working directory, use your course number. This tutorial may use a different course number as an example.
- The directory path shown in screenshots may be different from yours.
- If you are not sure what to do or confused with any steps:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

- Locate a suitable operating system based on this article here:
  - https://www.howtogeek.com/718012/how-to-choose-between-ubuntu-kubuntu-xubuntu-and-lubuntu/
  - After reading this article, your first decision (as a future engineer), is to look at your current HOST system (your laptop, desktop you will be working on in this course and answer the following questions:
    - 1.) Which of the four linux distributions would be the best fit for my current HOST resources. (If you need help, you can send the instructor an e-mail with the heading "how do I pick the best linux distribution image?" and then you will have to take a screenshot of your system properties like the one below:



- Download the image into a folder on your system.
  - Make sure you have enough disk space, some of the images are very large and will need to be extracted onto your host system. These files are disk images of an actual virtual machine.

- **Download and install Virtualbox and screenshot the version installed.**
- Import the Disk Image into Virtualbox.

Resources

- Obtain virtual machine disk images from Osboxes – https://www.osboxes.org/
- Obtain Virtualbox. At time of this writing, the download is at 6.1.18 – https://www.virtualbox.org/wiki/Downloads https://www.virtualbox.org/

Section 1: Your HOST Environment

Subection 1:1 ~ What is your system properties for your HOST?

By HOST, we refer to the system that will run a type-2 hypervisor. A type-2 hypervisor runs on top of an Operating System. Assumptions made:

a.) You are running Windows on a laptop or desktop.

b.) You will be installing Virtualbox (which makes this application your type-2 hypervisor).

Before we do that, we need to know your resources on your laptop or desktop. Unless otherwise stated, you can google how to find your system properties. We are only concerned with your current resources. As shown in a screenshot above, the HOST used for building this tutorial has ample amounts of processing and memory to build virtual machines. Your HOST system should have 16 GB of RAM and an i7 Intel processor to handle assigning multiple CPUs (2) and RAM (>4GB) for this virtual machine.

```
Your assignment task #1: Open a word document and capture a
screenshot (preferably with Snippet Tool that comes with Windows)
a picture of YOUR system properties and paste it into this
document. Label the Document HOP01.
```

Subsection 1:2 ~ Where to store your downloads?

Optional: Create a subfolder in your MyDocuments for this class IS340. You should store any files or screenshots for assignments in this folder for each of following future tutorials.

Section 2: Your Disk Image of Choice

Read the article mentioned above in the learning outcomes section. Based on your HOST machine's current resource values, answer this question and put this in the document HOP01:

Q: Why did you choose THIS image? What were some of the considerations made before selecting it?

Subsection 2:1 ~ How long does it take to download?

In our lab example, the author chose LinuxLite (https://www.osboxes.org/linux-lite/**Error! Hyperlink reference not valid.**

Your home download speed will vary. In this example, it took the author 30 minutes. Start the download and work on something else.

Section 3: Install Virtualbox

Subsection 3:1 ~ Which is the best version?

At the time of this writing, Virtualbox is at 6.1.18

Once installed, let's get familiar with two key areas you will be using a lot during this class:

1.) Tools ~

With Virtualbox, the Tools area is where you will Import, Export, add virtual machines. In our class, since we already have a pre-configured disk image (thanks to Osboxes), we can dispense with creating a new one from scratch and instead, create a new Virtual machine, but point the virtual disk drive to use that LinuxLite download.

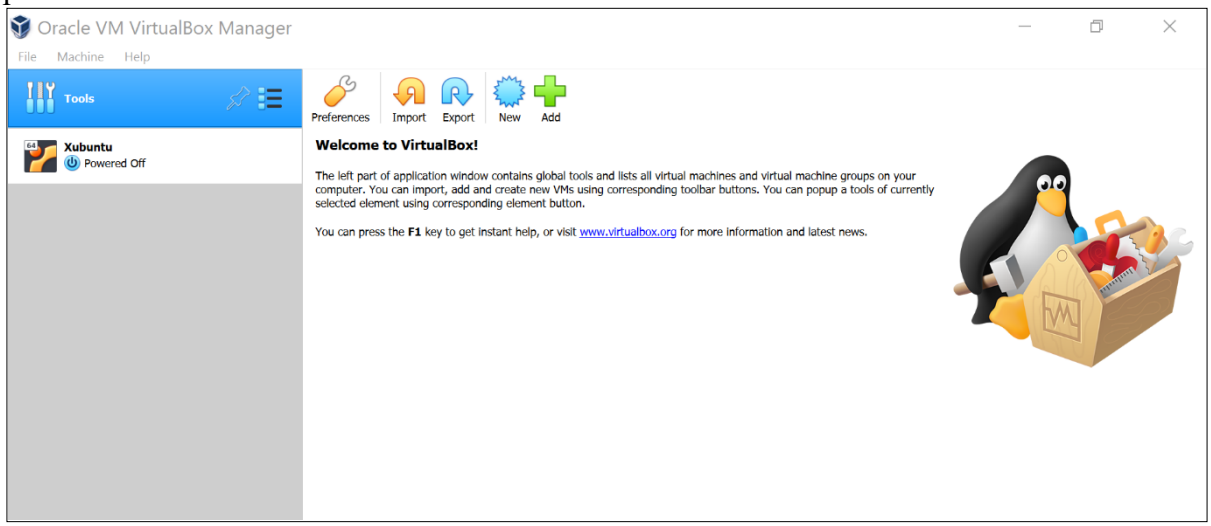
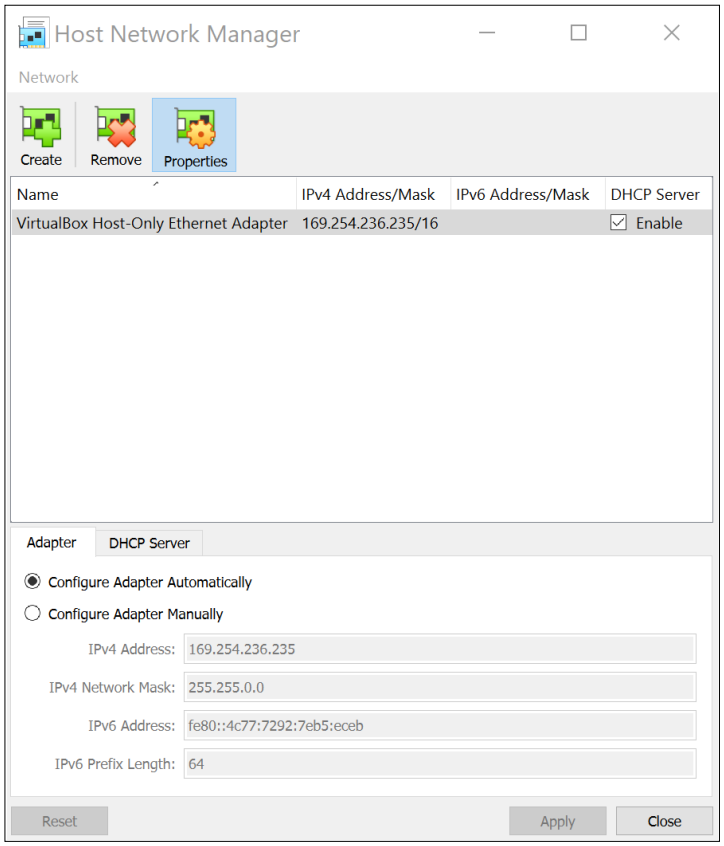


Fig. 3.1: Tools menu on Virtualbox 6.1.18

2.) HOST Network Manager ~

This is the place where you would create a virtual network to have virtual machines communicate with the HOST machine. This is extremely useful much later in this course and other courses where you would like your HOST machine to access some service you are running inside your virtual machine. This requires some basic knowledge of networking, but if you follow the tutorial, you should be fine. You can access the Host Network Manager by clicking File → Host Network Manager (or Cntrl + H)



Note: In a future lab, we will be making changes to this network.

Assignment task #2: Open Virtualbox and click on Help. Then screenshot the "About Virtualbox" window like the one below. Paste that into your HOP01 document you are working in.

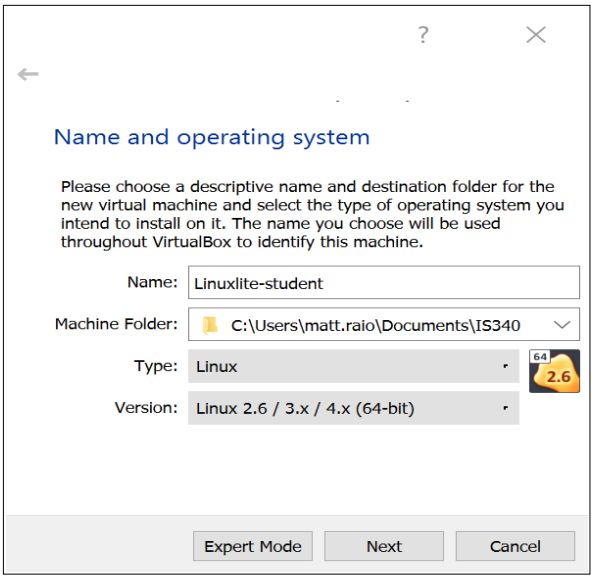
Section 4: Pulling it All Together

Subsection 4:1 ~ The Walk through:

Now that you have everything, perform the following tasks:

1.) Open Virtualbox.
2.) Make sure you click on the 'Tools' bar on the left.
3.) Select 'New' on the toolbar in the main screen.
4.) The name and operating system dialogue box is open.
For the "Name:" field you will put "LinuxLite-<insert your first name here>"
5.) Pick the sub folder you will be storing your image file in. The author chose to create a special sub folder called "IS340" in his Documents folder in Windows.

Note: Assignment 3: Screenshot your decisions here before clicking on "Next". Put this in your HOP01 document. ---------->



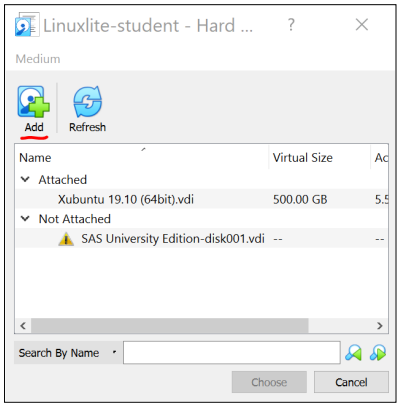
6.) The create virtual machine dialogue box is open.

Depending on your HOST machine determines what is a reasonable amount of memory to allocate. The default shows 1024 MB (or approx. 1 GB of memory), your HOST machine, should have enough to give it 2048 or 4096 MB. The more you can give the machine without impacting your host resources is a key design consideration. You can always adjust the memory later after the virtual machine is created. It is recommended to have at least 2048, 4095 is preferred.

Click "Next" (no screenshot required)

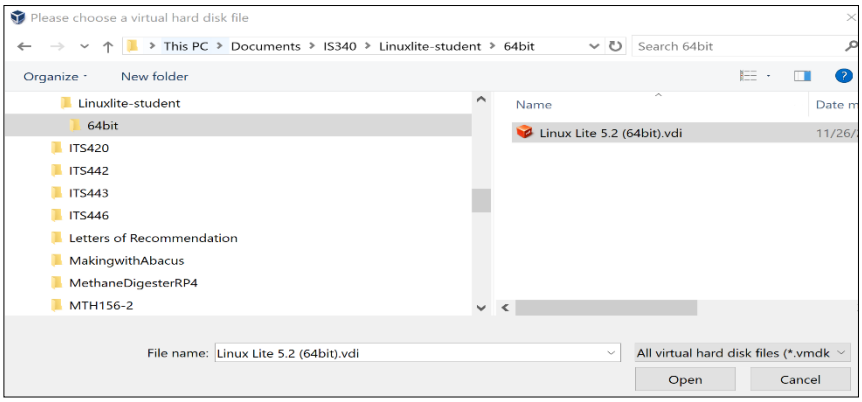
NOTE: You may have to extract your downloaded disk image into that new sub folder first if you haven't done that already.

7.) The next window is where you specify the disk image that was downloaded from Osboxes.



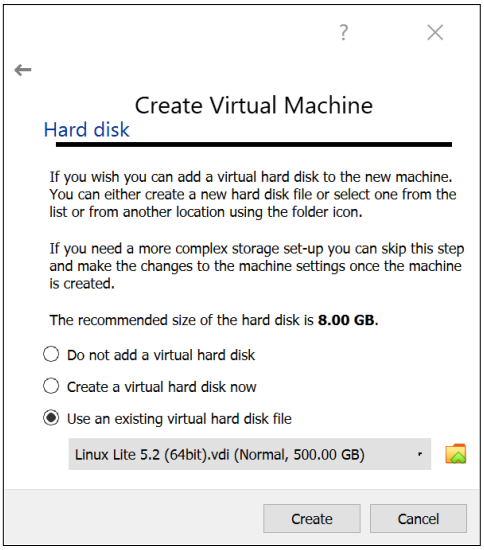
a.) Click 'Add'

b. Using the file browser window, locate the 'Linux Lite 5.2(64bit).vdi' Image as shown below:



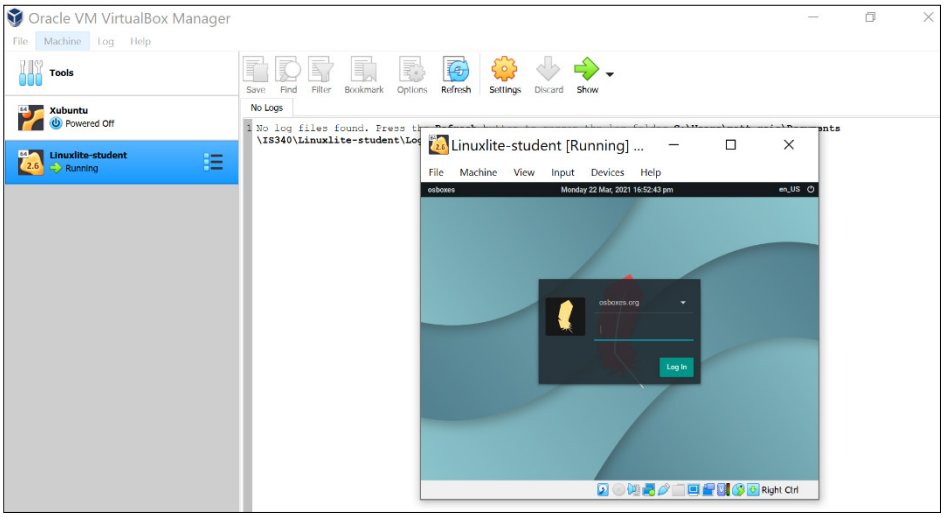
c. Select the image and click ‘Choose’.

d. Now click ‘Create’



Subsection 4:2 ~ Testing the Image Works…

Once you are at this point, you can now select your new image and power it on. You can start the image by click ‘Start’ on the tool bar or you can right click the virtual machine and select ‘Start’. Virtualbox will then try to select the best display based on your current HOST environment. The author has 3 different monitors and it may take some time adjusting the display to fit if Virtualbox doesn’t pick one that works for your HOST display settings.



Your assignment task #4: Get a screenshot showing your virtual machine is running. Looking for the name of the virtual machine (should have your student name in the naming of this virtual machine). Add this screenshot to your HOP01 document.

This completes the HOP assignment for Module 1. The next module will be about the networking aspect of using a virtual machine in Virtualbox.

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 Operating Systems

HOP02 – Basic Networking with Virtualbox's Host Network Manager

3/25/2021 Developed by Matt Raio
03/31/2021 Reviewed by Pengfei (Chuck) Liu
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- Version numbers may not match with the most current version at the time of writing. If given the option to choose between stable release (long-term support) or most recent, please choose the stable release rather than beta-testing version.
- This tutorial targets new users with limited experience with Linux.
- There might be subtle discrepancies along the steps. Please use your best judgment while going through this cookbook style tutorial to complete each step.
- For your working directory, use your course number. This tutorial may use a different course number as an example.
- The directory path shown in screenshots may be different from yours.
- If you are not sure what to do or confused with any steps:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

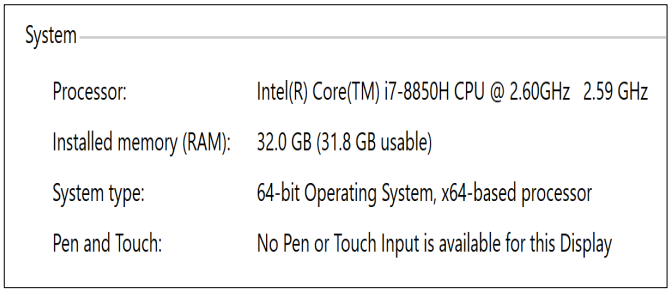
Learning Outcomes

Students will be able to:

- Perform basic network subnetting with the Host Network Manager through Virtualbox.
- Perform basic connectivity checks between the target virtual machine and the host machine.
- Understand the importance of diagramming (logically), the small network running inside a student HOST machine (laptop, desktop).

Prerequisite:

Please check your current system properties if running a Windows laptop/desktop. Also, based on the previous HOP00A, please be mindful of those resources taken for the virtual machine created.



Resources

The first HOP for Module 1 in IS340 should be completed before attempting this lab.

Purpose of this lab: At this point, you selected the IS340 Operating Systems to gain a better understanding of what an Operating System can do to integrate devices and improve user experience. In the last few years, tremendous progress has been made in the open source community to expand the role of the Operating System (OS) into key areas that would have taken entire IT Operations teams to manage. For example, code control and re-visioning was the task of IT Operations. Now Git, and Github perform code control and the OS now can integrate with access to these local and cloud resources.

In addition to extending the OS, in the last 15 years, the Operating System has now become a Type II hypervisor with a graphical overlay (GNS3). This single development was impactful for students who needed to build real virtual networks for study of certifications and create real transferable skills into the workplace. The author of this HOP, helped write for the online community when GNS3 was funded between the years of 2014-2016. This HOP will expand the role of the OS into areas either not previously thought of, or at the very least, open and challenge the student to use the OS in ways for growth and learning in your own "Proof of Concept" lab (or affectionately known as "a sandbox").

This particular lab hits at a weakness that ironically developers and even IT subject matter experts lack: network fundamentals. Consider this lab "a class within a class" aCwaC. What is expected from the student are screenshots at key points in this lab and uploaded into your BB account.

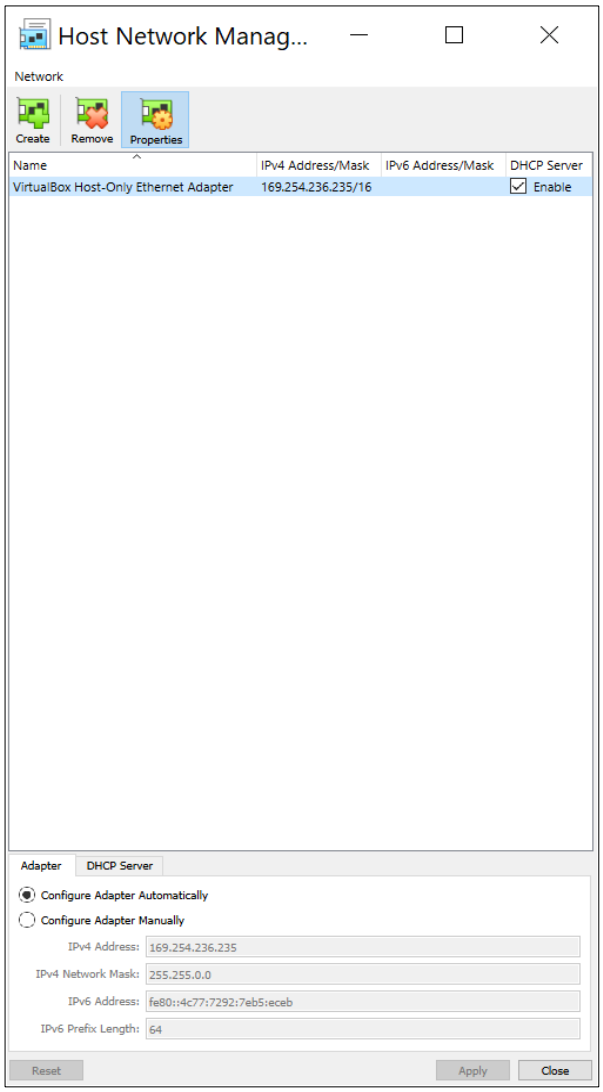
Section 1: Your Network Environment

Your operating system has changed forever after downloading and installing Virtualbox (or VMWare Workstation or VMWare Player). Why? Because in addition to building virtual machines, the OS becomes a Type II hypervisor. This now profoundly changes what the Operating System is going to do. When you installed the Virtualbox software, there is a little area called "Host Network Manager."

You will need to collect some information about the virtual network that Virtualbox uses to manage network access to the virtual machines and your HOST system (your laptop or desktop).

Section 1.1: Is Your Host Network Manager Configured?

From the main Virtualbox screen, select File → Host Network Manager



Now that you have created a default Host Network Manager, note the name of this manager as shown in the screenshot above. The example shows the name "Virtualbox Host-Only Ethernet Adapter", with a check box under the column "DHCP Enable" as checked.

This means, that when you create a new virtual machine, and then configure the network adapter inside this virtual machine to use THIS network, the Host Network Manager will run a DHCP service just for this network (herein referred commonly as a 'subnet') and your virtual machine will receive a 'local' IP address in the network of '169.254.236.x'. Where 'x' could be an IP address that is within the range of the DHCP to offer.

How do we know this? Click on the tab 'DHCP Server' toward the bottom of the screen shown above.

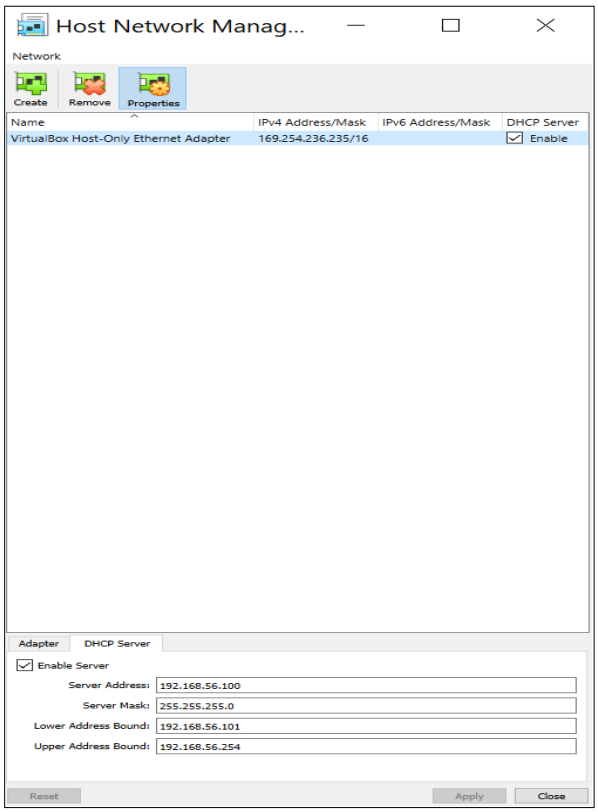
In this example, there is already a network defined within the HOST NETWORK MANAGER as shown above. If you don't have a network defined, click on the 'Create' button and Virtualbox will create a new network with defaults in place. In this example, what defaulted is a network 169.254.236.235/16.

Although this is not a networking course, it's important to understand what Virtualbox defaults created.

1. Question 1: How many hosts are available to use for virtual machines? If you are not sure, use an IP calculator: http://jodies.de/ipcalc.

Assignment Task #1: Enter the total number of possible hosts into your word document labeled "HOP02"

Section 1.2: What services does this Host Network Manager perform?



Section 1.2.1: The scope of your personal DHCP Server?

We can now test these settings. But first, let's define the scope the DHCP server is going to use:

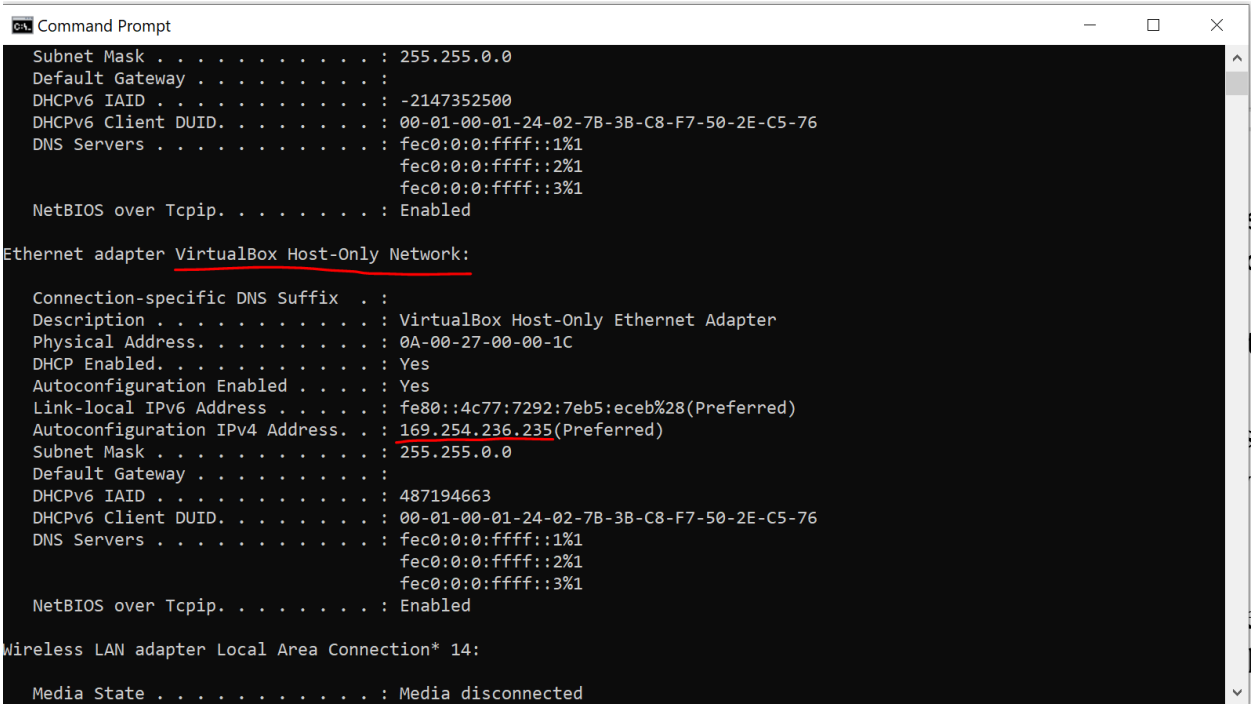
a.) The DHCP server's address is "192.168.56.100". The network mask for this network is "255.255.255.0" which in CIDR notation is /24. The range of host IP addresses start at "192.168.56.101 through 192.168.56.254."

Assignment Task #2: What is your network DHCP server address? What is the subnet mask? What are the ranges of possible hosts this instance of DHCP will use? Put this down in your document HOP02.

Section 2: Your HOST IP address for Virtualbox.

We will now check the IP address of your HOST machine. This address MUST be on the same subnet as both your instance of the DHCP AND the virtual machines you will configure in the future.

From windows open a command prompt and type "ipconfig /all". Depending on the number of network interfaces (physical network cards on the host system) AND the virtual ones in use by VMWare and Virtualbox, you need to focus on only that interface that is in the same subnet as your "VirtualBox Host-Only Ethernet Adapter"

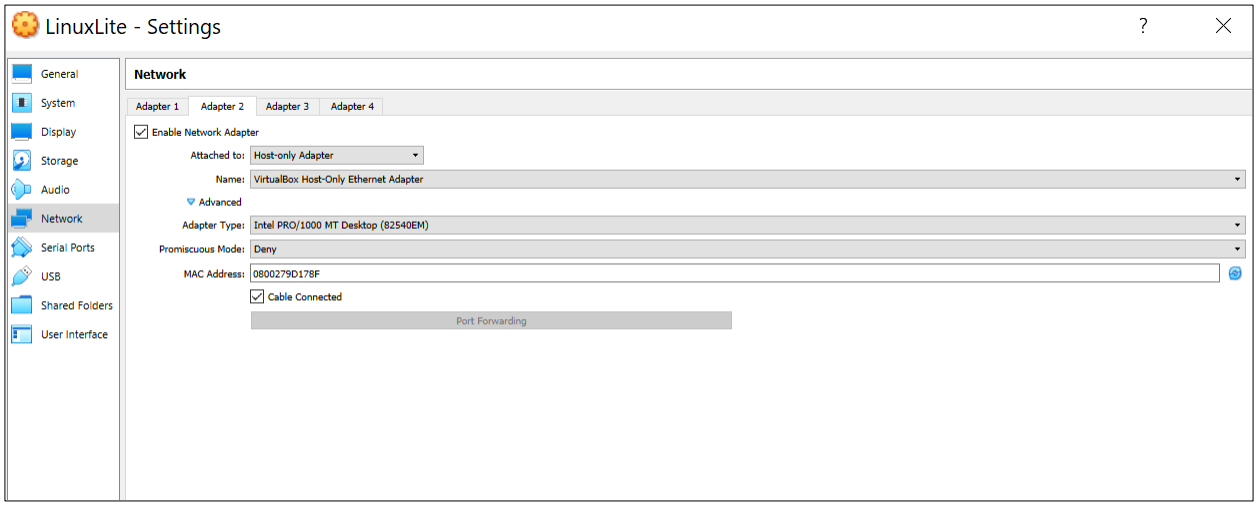


In this example, after running the "ipconfig all" command, a figurative stream of interfaces were displayed. Scroll up until you see the "VirtualBox Host-Only Network: adapter settings. In this example, the HOST's (your laptop) is configured with the address <169.254.235.235/16>*

Section 3: Add a new network interface to your virtual machine.

Step 1: Choose your virtual machine and click Settings.
Step 2: Select the "Network" link.
Step 3: Click on Adapter 2
Step 4: Click on the check box "Enable Network Adapter"
Step 5: Click on the drop down box and select "VirtualBox Host-Only Ethernet Adapter"
Step 6: Optional ~ Click Advanced and look at "Promiscuous Mode"

- This mode allows all traffic "broadcasts" and packet types to be allowed on this subnet.



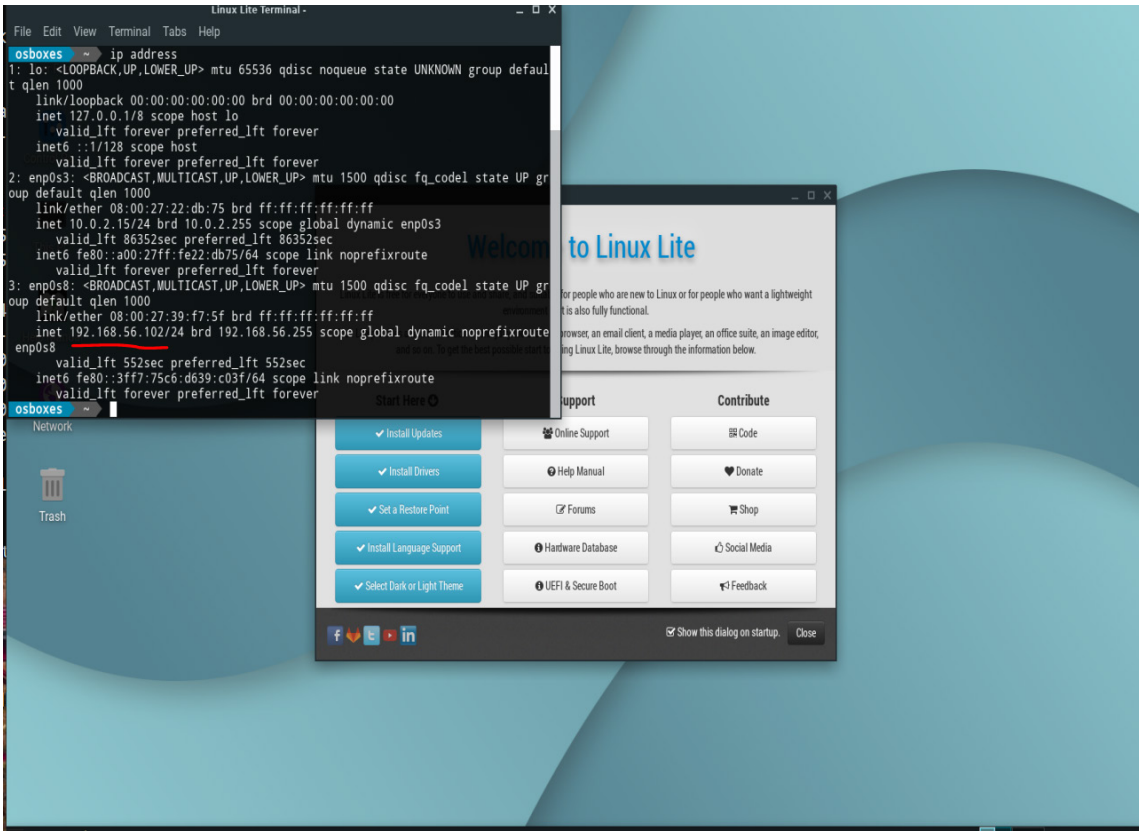
The result of this is your virtual machine has 2 network adapters. This is known as 'multi-homing' a device to have two networks available. The first adapter is by default a NAT adapter that uses your HOST machine to get out to the Internet. The second one now has connectivity to your local VirtualBox subnet as defined by the Host Network Manager in previous sections.

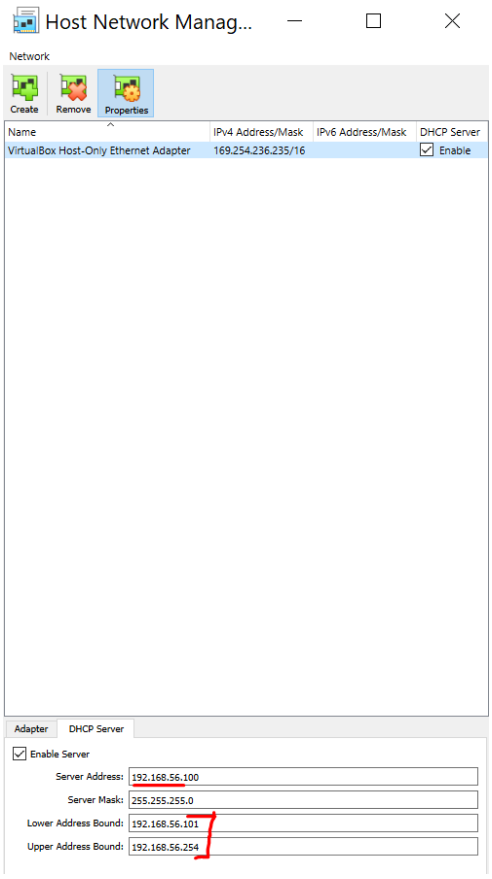
Section 4: Testing the connections between your HOST and the virtual machine(s).

Now that we have established the three endpoints (your first virtual machine, your HOST IP address and the "Virtualbox Host-Only Ethernet Adapter DHCP server", we can now do a test.

1.) Launch your virtual machine.
2.) Open a command prompt in the Linux shell.
3). Check if the DHCP has given your virtual machine an IP address.

- Open a terminal in the Linux vm and type "IP address"
- Similar to "ipconfig /all" in windows, the OS returns what interfaces are configured (if any)





The above illustration shows a new interface called "enp0s8" with an address of "192.168.56.102". Does this address fit in the scope of the DHCP server? Check one more time to make sure.

We can see that the vm host address of 192.168.56.102 fits inside this scope. There is a good reason to believe that your virtual machine requested a lease and the DHCP instance inside the "VirtualBox Host-Only Ethernet Adapter" responded and issued it. NOTE: This is assuming you did not set a 'static' IP address with the same value.

4.) Use the PING tool to get a reply back from the HOST machine.

From your HOST machine, open a command prompt and type "PING <192.168.56.102>*"

```
Pinging 192.168.56.102 with 32 bytes of data:
Reply from 192.168.56.102: bytes=32 time<1ms TTL=64
Reply from 192.168.56.102: bytes=32 time<1ms TTL=64
Reply from 192.168.56.102: bytes=32 time<1ms TTL=64
Reply from 192.168.56.102: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.102:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

A response from some client shows that your HOST machine has connectivity to communicate with your new virtual machine!

NOTE: To ping from the virtual machine to the HOST requires a completely different network adapter type BRIDGED. This is beyond the scope of allowing your HOST machine access to the virtual machine in this class.

Assignment Task #3: Logically represent in a simple diagram the three elements in your first network topology. It doesn't have to be fancy, you can use whatever tool (MSPAINT, Lucidcharts, Visio) that you are comfortable with. But the endpoints have to be labeled with the following:

1.) Name of the device, the IP address and subnet mask (you can use the cidr / notation.

2.) Draw a shape for each entity and draw lines to show how they are connecting.

3.) Put all this in the same document for future use / reference.

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 – Operating Systems
HOP03 – BASH – Your First Shell Program
07/10/2019 Developed by Kevin Wang
07/12/2019 Reviewed by Clark Ngo and Bill Kaghan
03/20/2020 Reviewed by Kim Nguyen
10/03/2020 Reviewed by Kim Nguyen
03/29/2021 Updated by Matt Raio
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- You may use the new ubuntu based linux distribution (LinuxLite) OR if you have taken another CityU class, your EC2 instance. Connecting to your EC2 class should already be covered in the class you used the EC2 instance.

- All commands and code discussed in this exercise will run in the Ubuntu console.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself. (The tutorial will provide reminders.)
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes
Students will be able to:

- Create a script file
- Modify file permissions
- Execute echo and printf commands
- Run a bash script

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/
  (You can use this reference to find descriptions of all the Bash commands that you will use in this and future hands-on exercises. It is recommended that you consult this reference frequently until you are familiar with frequently used commands.)

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

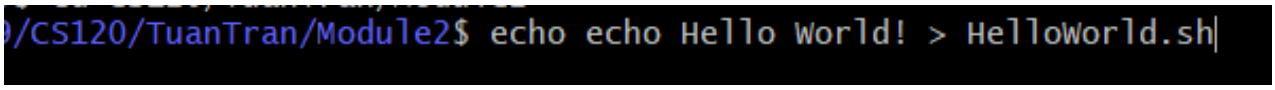
(NOTE: Your directory and file names might be different from screenshots, as they are subject to be used for a different course)

Preparation

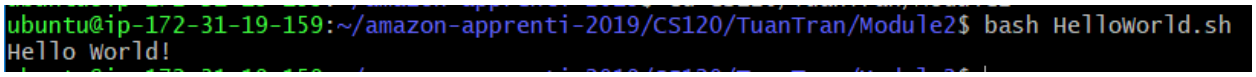
Connect to your Ubuntu instance

Creating, saving, and running the first Bash Script

1) Create your first script file by typing the following command in the console command line:
   >>> echo echo Hello World! > HelloWorld.sh
   Note:
   We are using the > to send the output of echo command to a file called HelloWorld.sh



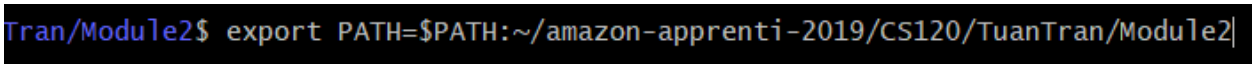
2) Run the script file by typing follow command:
   >>> bash HelloWorld.sh



3) Add your directory to the path:

>>> export PATH=$PATH: ~/IS340/HOP03-YourGithubUsername/Module 3

[NOTE: the path above is an example, you need to replace the path with your actual Github repo]

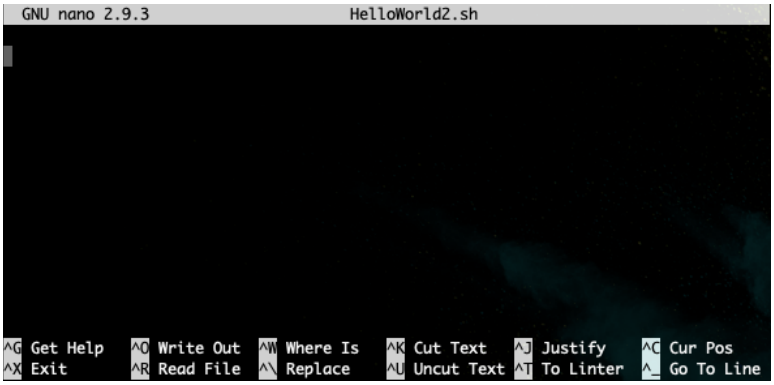


4) Give the execution permission to your file:
   >>> chmod +x HelloWorld.sh

5) Now we can execute the file without typing the bash command:
   >>> HelloWorld.sh

Use nano to edit the file

1) Type the following command to create a file with the nano editor:
   >>> nano HelloWorld2.sh



You should see an editor UI like this

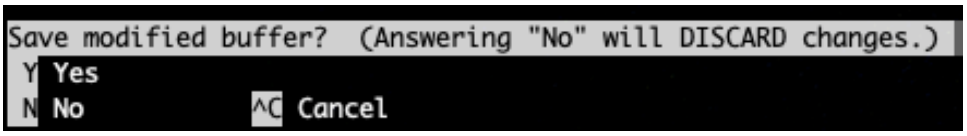
2) Type the following commands in the editor:

```
#! /bin/bash

printf "%s\n" "Hello World!"
```

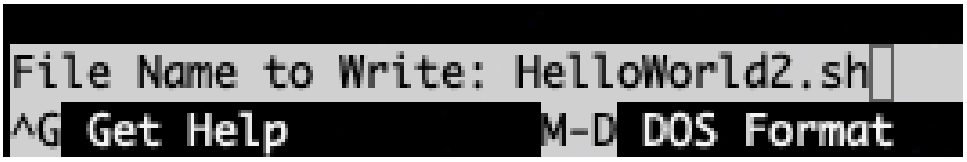
The first line is a shebang, which tells the shell what program to interpret the script with, when executed. In this example, the script is to be interpreted and run by the bash shell.

For the second line, the first part "%s\n" define how to print the text (%s means printing as string. \n means print a line break after the text)
The second part is the actual text that we want to print.
More format interpterion can be found from https://wiki-dev.bash-hackers.org/commands/builtin/printf

3) Hit the control + x key to quit the editor. You will see as below:



Then hit y key to confirm. The following message will appear as shown below:



Hit the enter key to save the changes

4) Run the script by typing:
   >>> bash HelloWorld2.sh

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 – Operating Systems
HOP04 - Input, Output and Throughput
7/13/2019 Developed by Kevin Wang
06/17/2020 Reviewed by Kim Nguyen
03/29/2021 Updated by Matt Raio
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- This exercise assumes that the user is working with a Linux virtual machine in virtualbox.

- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes
Students will be able to:

- Use parameters and variables
- Format and print data with printf command
- Use input/Output, pipeline, and command substitution

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

Preparation
1) Connect to your Ubuntu instance

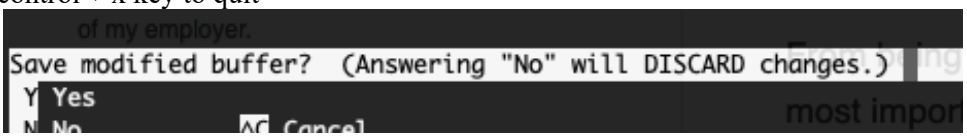
Using parameters and variables
1) Type the following command to reuse HelloWorld.sh file:
   >>> nano HelloWorld.sh

2) Update the file content as below:

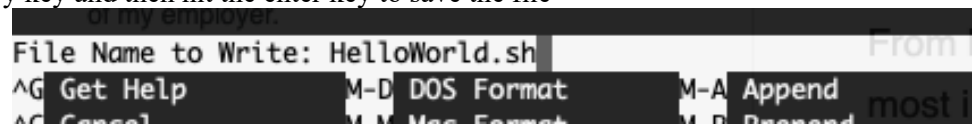
```
#! /bin/bash
printf "Hello %s!\n" "$1"
printf "Value of all the positional parameters: %s\n" "$*"
printf "The number of positional parameters: %s\n" "$#"
printf "The name of the current running script: %s\n" "$0"
printf "The process identification number (PID): %s\n" "$$"
printf "The exit code of the last-executed command: %s\n" "$?"

# Define a varialbe and use it
newName="Arthur Wang"
printf "I have a cool name which is %s\n" "$newName"
```

Note: A positional parameter is an argument specified on the command line. Positional parameter values are stored in a special set of variables that can be accessed via positional parameters. For example, the first, second, and third positional parameter values can be retrieved via the positional parameter $1, $2, and $3, respectively. Parameters greater than 9 can be accessed by using curly braces around the number; for instance, ${10} would be the tenth parameter.
Besides positional parameters, there are other special parameters to retrieve other information, such as PID and exit code of the last executed command, as shown in the first part of the script.

3) Click the control + x key to quit



4) Type the y key and then hit the enter key to save the file



5) Type the following command to see the result (please change the name to your name)
Note: the command below has 3 parts: interpreter (i.e. bash shell); command name (i.e. HelloWorld.sh); and arguments (i.e. Kevin Wang).
>>> bash HelloWorld.sh **YourName**

```
ubuntu@ip-172-31-25-112:~/CS120/Module2$ bash HelloWorld.sh Kevin Wang
Hello Kevin!
Value of all the positional parameters: Kevin Wang
The number of positional parameters: 2
The path to the current running script: HelloWorld.sh
The process identification number (PID): 15611
The exit code of the last-executed command: 0
I have a cool name which is Arthur Wang
```

Format and print data with printf command
The basic syntax for printf command is: **printf format arg1 arg2 …**
You can find all escape codes and format specifiers here: https://wiki-dev.bash-hackers.org/commands/builtin/printf

1) Create a TestPrintf.sh by typing the following command in the console command line:
   >>> nano TestPrintf.sh

2) Type scripts in the file as below:

#! /bin/bash
printf "Two tabs after me\t\tA new line after me\nThe hexadecimal digits X42 represents \x42\n"

printf "%s\n" Print in separated lines "Print in one line"
printf "%b\n" "%b tells printf to escape sequences. Two tabs after me\t\t ."
printf "%s%d\n%s%f\n%s%e\n" "%d prints integer such as: " 25 "%f prints float numbers such as: " 25.5 \ "%e prints with the exponential notation such as: " 25. 5
printf "%s#%02x%02x%02x\n" "%x prints in hexadecimal. Here is an example to convert a RGB color 82 185 225 to a hex notation: " \ 82 185 255

#Use width specification
header="\n %-10s %-18s %8s\n"
format=" %-10d %-18s %8.2f\n"
printf "$header" Id Name "Order price"
printf "==================================\n"
printf "$format" 1 "Kevin Wang" 234.30 2 "Arthur B" 332.23 3 "Evan A" 525.32

```
#! /bin/bash
printf "Two tabs after me\t\tA new line after me\nThe hexadecimal digits X42 represents \x42\n"

printf "%s\n" Print in separated limes "Print in one line"
printf "%b\n" "%b tells printf to escape sequences. Two tabs after me\t\t ."
printf "%s%d\n%s%f\n%s%e\n" "%d prints integer such as: " 25 "%f prints float numbers such as: " 25.5 \
       "%e prints with the exponential notation such as: " 25.5
printf "%s#%02x%02x%02x\n" "%x prints in hexadecimal. Here is an example to convert a RGB color 82 185 225 to a hex notation: " \
       82 185 255

#Use width specification
header="\n %-10s %-18s %8s\n"
format=" %-10d %-18s %8.2f\n"
printf "$header" Id Name "Order price"
printf "==========================================\n"
printf "$format" 1 "Kevin Wang" 234.30 2 "Arthur B" 332.23 3 "Evan A" 525.32
```

3) Hit the control + x key to quit and save the file

4) Execute the file to see the result by typing following command:
>>> bash TestPrintf.sh

Input and Output

1) Create a file by using the standard output operation >
Type the following command in the console command line:
>>> printf "%s\n" "My text in the file" > test

2) Check the file content by typing the following command:
>>> cat test

```
ubuntu@ip-172-31-25-112:~/CS120/Module2$ printf "%s\n" "My text in the file" > test
ubuntu@ip-172-31-25-112:~/CS120/Module2$ cat test
My text in the file
```

3) Read data from a file to a variable by using the standard input operation <
>>> read content < test

4) Check the variable value by typing the following command:
>>> echo $content

```
ubuntu@ip-172-31-25-112:~/CS120/Module2$ read content < test
ubuntu@ip-172-31-25-112:~/CS120/Module2$ echo $content
My text in the file
```

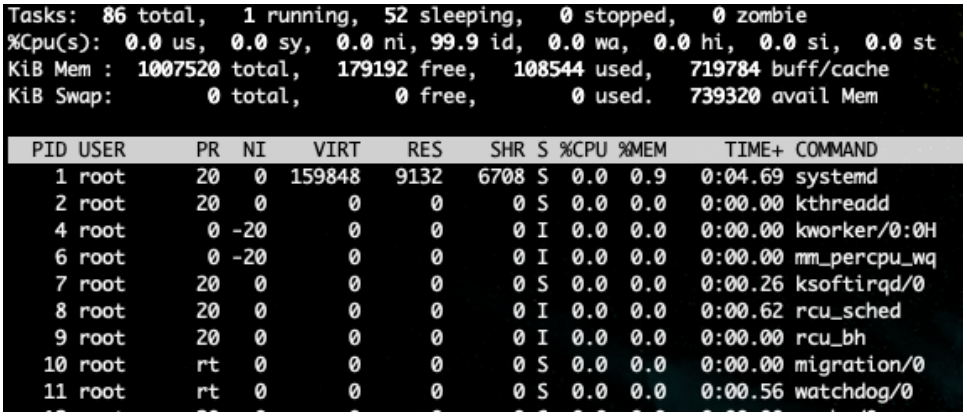
Using pipelines and command substitution

A pipeline can help users to take one command's output as another command's input

1) Save top command result to a file by typing the following command in the console command line:
>>> top -n 1 | tee result
You will see the result from top command

2) Clear the screen by typing the following command:
>>> clear

3) Check the result file:
>>> cat result



4) We also can use the command substitution to store the command output to a variable:
   >>> lsResult=$(ls)
   We are saving the ls command's output to the lsResult variable

5) Check the variable value by typing the bash command:
   >>> echo $lsResult

Take a screenshot of your printf commands as well as your pipelines and substitutions.

What commands did you have to substitute? Were there any challenges you faced with the instructions vs. what your OS had in its environment?

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 – Operating Systems
HOP05 – BASH – Looping and branching
7/13/2019 Developed by Kevin Wang
06/17/2020 Reviewed by Kim Nguyen
03/29/2021 Updated by Matt Raio
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- This exercise assumes that the user is working with the Linux distribution in Virtualbox

- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes
Students will be able to:

- Use test command
- Understand the conditional execution
- Use input/Output, pipeline, and command substitution

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

Preparation

1) Connect to your Ubuntu instance

Basic usage of the test command

1) Type the following command to test integers:
   >>> test 1 -eq 1

Note: -eq for equal, -ne for not equal, -gt for greater than, -lt for less than, -ge for greater or equal, -le for less or equal.

2) Check the result by typing:
>>> echo $?
Note: $? is a special parameter that contains the exit code for the last command. 0 means successful when 1 means error. When we use a test command, 0 means true and 1 means false.

```
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ test 1 -eq 1
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ echo $?
0
```

3) [] is the equivalent of test. We can test an integer like this:
>>> [1 -lt -2]
And then, retrieve the result:
>>> echo $?

```
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ [ 1 -lt -2 ]
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ echo $?
1
```

Note: Linux Bash script is case sensitive as well as space sensitive. For using the [], we have to keep a space after [and before].

4) Use [[]] to test a regular expression
>>> [[abc =~ ^a.*c$]]
And retrieve the result
>>> echo $?

```
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ [[ abc =~ ^a.*c ]]
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ echo $?
0
```

Note: We will practice more regular expression in other modules.

5) Use (()) to evaluate an arithmetic expression. (This is a nonstandard feature)
>>> ((2 - 1))
>>> echo $?
>>> ((1 - 1))
>>> echo $?

```
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ (( 2 -1 ))
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ echo $?
0
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ (( 1 - 1 ))
ubuntu@ip-172-31-19-159:~/amazon-apprenti-2019/CS120/TuanTran/Module6$ echo $?
1
```

In Linux, when arithmetic expression value is 0, it will return False. Otherwise, a true will be returned.

If conditional execution

1) The basic syntax of if command shows below:

if <condition list>
then

<list>
elif <condition list>
then
<list>
else
<list>
fi

2) Create a ConditionTest.sh file by typing following command:
>>> nano ConditionTest.sh

3) Type the following script in the file:

```
#! /bin/bash
printf "How many pizzas do you want to buy?\n"
read amount

if (( amount <= 0 ))
then
   printf "Please enter a number that is greater than 0.\n"
elif (( amount > 10 ))
then
   printf "Sorry, we do not have that many pizzas.\n"
   exit 1
else
   printf "There you go. Here are %d pizzas for you.\n" "$amount"
fi
```

4) Hit the control + x key to quit and save the file.

5) Type the following command to execute the script:
>>> bash ConditionTest.sh
Then type in a number to test it. Please run the program several times to try different numbers.

Case condition execution

1) The basic syntax for "case":

case WORD in
PATTERN) COMMANDS ;;
PATTERN) COMMANDS ;; ## optional
esac

2) Create a CaseTest.sh file by typing the following command:
>>> nano CaseTest.sh

3) Type the following script in the file:

```
#! /bin/bash
case $1 in
    *[!0-9]*) message="non-numeric character(s).";;
    *) message="a number.";;
esac

printf "You entered %s\n" "$message"
```

4) Hit the control + x key to quit and save the file.

5) Type the following command to see the result
>>> bash CaseTest.sh 10
>>> bash CaseTest.sh string

Looping

1) Type the following command to create a WhileTest.sh file:
>>> nano WhileTest.sh

2) Type the following script in the file:

```
#! /bin/bash
n=1;
while [ $n -le $1 ]
do
  printf "The loop has already run %d times.\n" "$n"
  n=$(( $n + 1 ))
done
```

3) Click the control + x key to quit and save

4) Type the following command to test the file. You can give a different number to the parameter.
>>> bash WhileTest.sh 10

5) An Until loop is the opposite of while loop, which will run the code as long as the condition fails.
Type the following command to create a UntilTest.sh file:
>>> nano UntilTest.sh

```
#! /bin/bash
n=1;
until [ $n -gt $1 ]
do
  printf "The loop has already run %d times.\n" "$n"
  n=$(( $n + 1 ))
done
```

6) Click the control + x key to quit and save.

7) Type the following command to test the file:
>>> bash UntilTest.sh 5

8) Another common loop is for loop. Type the following command to create a ForTest.sh file:
>>> nano ForTest.sh

9) Type the following script in the file:

```
#! /bin/bash
# loop in different strings
for name in Kevin Arthur Evan
do
  printf "%s\n" "$name"
done

# loop in numbers (C likes syntax)
for (( i=0; i<= 10; i++ ))
do
  printf "The loop runs %d times.\n" "$i"
done
```

10) Click the control + x key to quit and save.

11) Type the following command to test the file:
>>> bash ForTest.sh

Take substitute variables for a random number and run the ConditionTest.sh again. Take a screenshot.

Change the condition in ForTest loop to output a different print statement that shows how many times the loop ran. Take a screenshot.

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 – Operating Systems
HOP06 – BASH – Command-Line Parsing and Expansion
7/14/2019 Developed by Kevin Wang
6/17/2020 Reviewed by Kim Nguyen
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- This exercise assumes that the user is working with the Linux Distribution in MODS 1 and MOD 2
- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

- Parse command line with quoting, brace, tilde, variable, and arithmetic expansions
- Understand command substitution, word splitting, and pathname expansion
- Catch parsing options

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

Preparation

1) Create a new script via the command below:
   >>> nano ShowArgument.sh
   Then, type the following script to the file:

```
#! /bin/bash
index=1
for arg in "$@"
do
  printf "Argument %d: %s\n" "$index" "$arg"
  index=$(( $index + 1 ))
done
```

Note:
This script can print out different arguments followed by a command, which can help us to see how the bash script parse the arguments.

2) Hit the control + x key to quit and save the file.

3) Run a test by typing the following command:
>>> bash ShowArguments.sh first second 'third argument'
You are supposed to see the following result

```
Argument 1: first
Argument 2: second
Argument 3: third argument
```

Note: please make sure you run the script successfully and get the correct output since it will be used for the next step.

Expansions

1) Testing the quoting style for command arguments by typing the following command:
>>> bash ShowArguments.sh argument1 argument2 'I am argument 3'
>>> bash ShowArguments.sh 'This is how to include "double quotes"'
>>> bash ShowArguments.sh "This is how to include 'single quotes'"

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh argument1 argument2 'I am argument 3'
Argument 1: argument1
Argument 2: argument2
Argument 3: I am argument 3
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh 'This is how to include "double quotes"'
Argument 1: This is how to include "double quotes"
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh "This is how to include 'single quotes'"
Argument 1: This is how to include 'single quotes'
```

Note: please type the command by yourself since Word uses different characters for quotes.

2) Testing the brace expansion by typing the following commands:
>>> bash ShowArguments.sh {first,second,third}

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh {first,second,third}
Argument 1: first
Argument 2: second
Argument 3: third
```

>>> bash ShowArguments.sh {{1..5},{a..d}}

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh {{1..5},{a..d}}
Argument 1: 1
Argument 2: 2
Argument 3: 3
Argument 4: 4
Argument 5: 5
Argument 6: a
Argument 7: b
Argument 8: c
Argument 9: d
```

>>> bash ShowArguments.sh {1..3}{a..b}

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh {1..3}{a..b}
Argument 1: 1a
Argument 2: 1b
Argument 3: 2a
Argument 4: 2b
Argument 5: 3a
Argument 6: 3b
```

>>> bash ShowArguments.sh {01..10..2}

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh {01..10..2}
Argument 1: 01
Argument 2: 03
Argument 3: 05
Argument 4: 07
Argument 5: 09
```

>>> bash ShowArguments.sh {A..G..2}

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh {A..G..2}
Argument 1: A
Argument 2: C
Argument 3: E
Argument 4: G
```

3) Testing the tilde expansion by typing the following commands:
>>> bash ShowArguments.sh ~
>>> bash ShowArguments.sh ~root

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh ~
Argument 1: /home/ubuntu
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh ~root
Argument 1: /root
```

Note: using the ~ + user's name to get the users' home directory. The username in this example is ubuntu.

4) Testing the variable expansion by typing the following commands:
>>> name=yourname
>>> bash ShowArguments.sh "${name}"

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ name=Kevin
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh "${name}"
Argument 1: Kevin
```

5) Testing the arithmetic expansion by typing the following commands:
>>> bash ShowArguments.sh “$((1 + 2))” “$(((1 + 2) * 2))”

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh "$(( 1 + 2 ))" "$(( (1 + 2) * 2 ))"
Argument 1: 3
Argument 2: 6
```

Understand command substitution, word splitting, and pathname expansion

1) Command substitution is used to replace a command with its output. Testing it by typing the following commands:
>>> echo "Today is $(date +%Y-%m-%d)" > file.md
>>> cat file.md

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ echo "Today is $( date +%Y-%m-%d )" > file.md
ubuntu@ip-172-31-20-124:~/CS120/Module4$ cat file.md
Today is 2019-07-15
```

2) The parameter will be split if we pass it as a variable without quotes. Test it by typing the following commands:
>>> names=”Kevin Arthur Evan”
>>> bash ShowArgurments.sh $names
>>> bash ShowArgurments.sh “$names”

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ names="Kevin Arthur Evan"
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh $names
Argument 1: Kevin
Argument 2: Arthur
Argument 3: Evan
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh "$names"
Argument 1: Kevin Arthur Evan
```

Note: You can see the difference when we use double quotes for the variable.

3) If the command has a parameter that contains *, ?, and [], bash will treat it as file patters. Testing it by typing the following commands:
>>> bash ShowArguments.sh Show*
>>> bash ShowArguments.sh ?ile.md
>>> bash ShowArguments.sh [a-f]ile*

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh Show*
Argument 1: ShowArguments.sh
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh ?ile.md
Argument 1: file.md
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh [a-f]ile*
Argument 1: file.md
```

4) The process substitution can be useful when we want to get the temporary file name for a command. Typing the following command to test:
>>> bash ShowArguments.sh <(ls -l) >(pr -Tn)

```
ubuntu@ip-172-31-20-124:~/CS120/Module4$ bash ShowArguments.sh <(ls -l) >(pr -Tn)
Argument 1: /dev/fd/63
Argument 2: /dev/fd/62
```

Note: this command allows the script getting a temporary input and output filename, which can allow programmers to read or write to.

Parsing Options

Bash script has a built-in command called getopts that can be used to parse options.
The basic syntax is: getopts OPTSTRING var

1) Create a OptionTest.sh file by typing the following command:
   >>> nano OptionTest.sh

2) Type the following script in the file:

```
#! /bin/bash
flags=
while getopts "a;v;c" opt
do
  case $opt in
      a) flags="a";;
      v) flags="v";;
      c) flags="c";;
      *) flags="invalid";;
  esac
done

printf "The flag is %s\n" "$flags"
```

Note: we accept a, v, and c as legal flags. All other flags will be seen as invalid flags.

3) Hit the control + x key to quit and save

4) Test the script with the following commands:
   >>> bash OptionTest.sh -a
   >>> bash OptionTest.sh -v
   >>> bash OptionTest.sh -c
   >>> bash OptionTest.sh -b

```
ubuntu@ip-172-31-25-112:~/CS120/Module4$ bash OptionTest.sh -a
The flag is a
ubuntu@ip-172-31-25-112:~/CS120/Module4$ bash OptionTest.sh -v
The flag is v
ubuntu@ip-172-31-25-112:~/CS120/Module4$ bash OptionTest.sh -c
The flag is c
ubuntu@ip-172-31-25-112:~/CS120/Module4$ bash OptionTest.sh -b
OptionTest.sh: illegal option -- b
The flag is invalid
```

Instructions:

Change values like the range of integers in the first example of ShowArguments.sh. Print a screenshot and upload that into your document.

Change the name values in ShowArguments.sh. Print a screenshot and upload that into your document.

Change the name of the file in the ShowArguments.sh. Print a screenshot and upload that into your document.

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 Operating Systems
HOP07 – Installing Packages On Your Linux VM

3/25/2021 Developed by Matt Raio
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- Version numbers may not match with the most current version at the time of writing. If given the option to choose between stable release (long-term support) or most recent, please choose the stable release rather than beta-testing version.
- This tutorial targets new users with limited experience with Linux.
- There might be subtle discrepancies along the steps. Please use your best judgment while going through this cookbook style tutorial to complete each step.
- For your working directory, use your course number. This tutorial may use a different course number as an example.
- The directory path shown in screenshots may be different from yours.
- If you are not sure what to do or confused with any steps:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

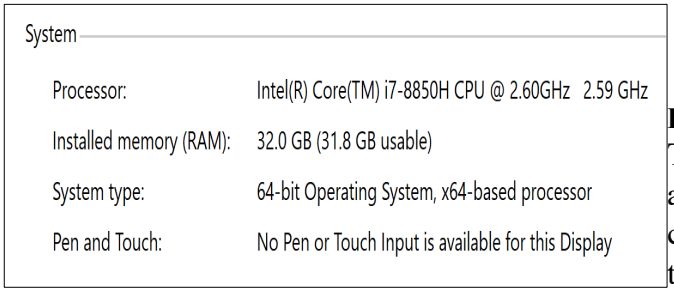
Learning Outcomes

Students will be able to:

- Install packages that extend the Linux OS to serve the needs of the student.

Prerequisite:

Please check your current system properties if running a Windows laptop/desktop. Also, based on the previous HOP01, please be mindful of those resources taken for the virtual machine created.



Resources

The first HOP for Module 1 and 2 in IS340 should be completed before attempting this lab.

Purpose of this lab: At this point, you have accomplished much in the sheer volume of HOPS and VL covered by the time you get to this HOP in week 7. Having said that, we can assume the following:

**Time may vary in download speeds. *Your address most likely will be different.

The student has completed to his or her ability all HOPS (1 – 6) and has documented this and uploaded it into BB for grading. It's important since the student is being graded not necessarily for correctness of a given task, but that it was documented for completeness.

Identifying one's weaknesses is critical in identifying those tasks that need to be worked on in your personal sandbox. The sandbox is a personal environment to try your hand at building an exact working model of whatever it is you wish to specialize in or support.

Consequently, the Linux virtual machine should be built and added to based on ***your*** academic and professional career. This machine will get built up, or torn down due to limitations of the HOST machine or quite frankly, the student may wish to go another route.

For example, you may take this course to get an understanding of the OS, but what is the application for you personally? Either a student follows the course of a developer, a network engineer or some combination of that as a DevOps engineer. With that, this particular module is basically the same with the exception that the student has the option to install a package he or she believes will be better suited for each individual goal.

Consequently, this HOP is about one of two options:

1.) Follow this tutorial to install PyCharm (a python IDE) on your linux vm.

2.) If you don't want to use PyCharm, or you want to install a different package from the command line, then answer the following question:

```
Assignment Task #1: Are you going to follow the tutorial and install
the recommended package? If not, what is the package you want to
install? Write this in a document labeled HOP07.
```

Section 1: Ways to install packages

Generally speaking there are three primary ways to install packages.

- Use a package manager like the Advanced Package Tool (apt) from the command line.
- Use a graphical package manager like Synaptic Package Manager.
- Follow the instructions to install a compressed file and uncompress it and run the installer script.
- Build your own from source code.

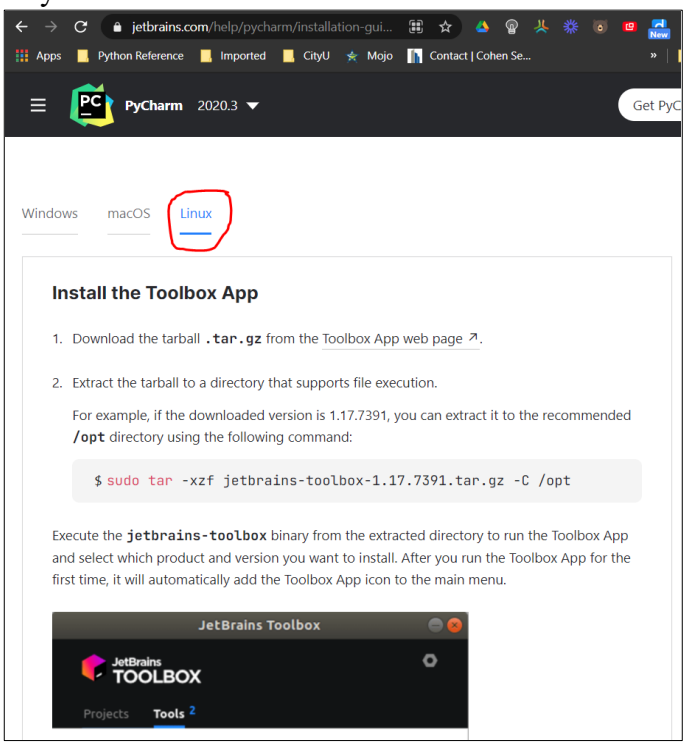
**Time may vary in download speeds. *Your address most likely will be different.

Since this is an essentials course, we will focus only on the first two options. PyCharm has a way to install the package directly from the website because it calls a separate routine to install on your system. But generally, the preferred method is to get used to using the command line (apt) since there are many HOW-Tos on different websites that you can literally copy and paste the command into the terminal if the syntax is causing errors at time of execution.

Section 2: Some HOW-TO guides are better than others…

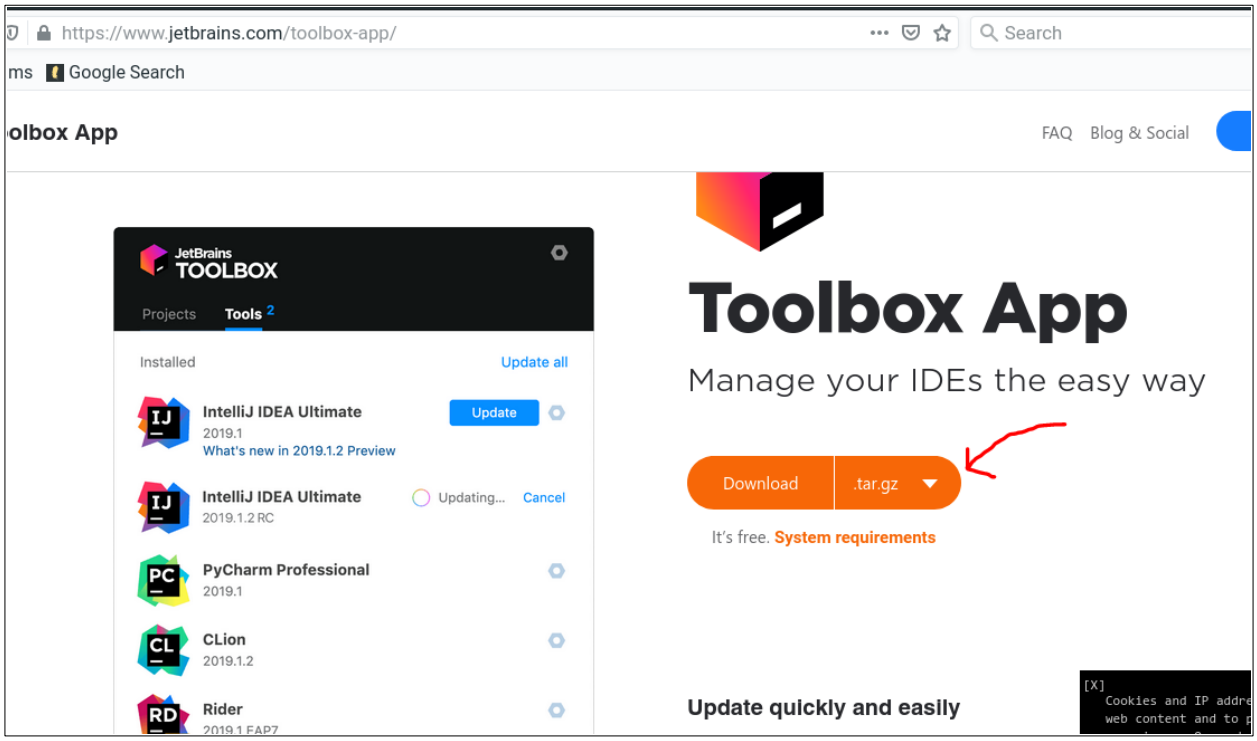
Go to this website to download the PyCharms compressed installer:
https://www.jetbrains.com/help/pycharm/installation-guide.html#toolbox

There is a 'Linux' tab you need to click on.



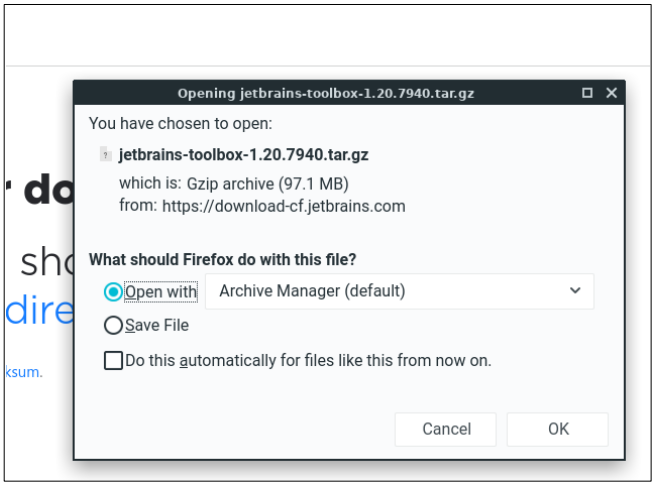
<------Click on the hyperlink "Toolbox App web page"

**Time may vary in download speeds. *Your address most likely will be different.



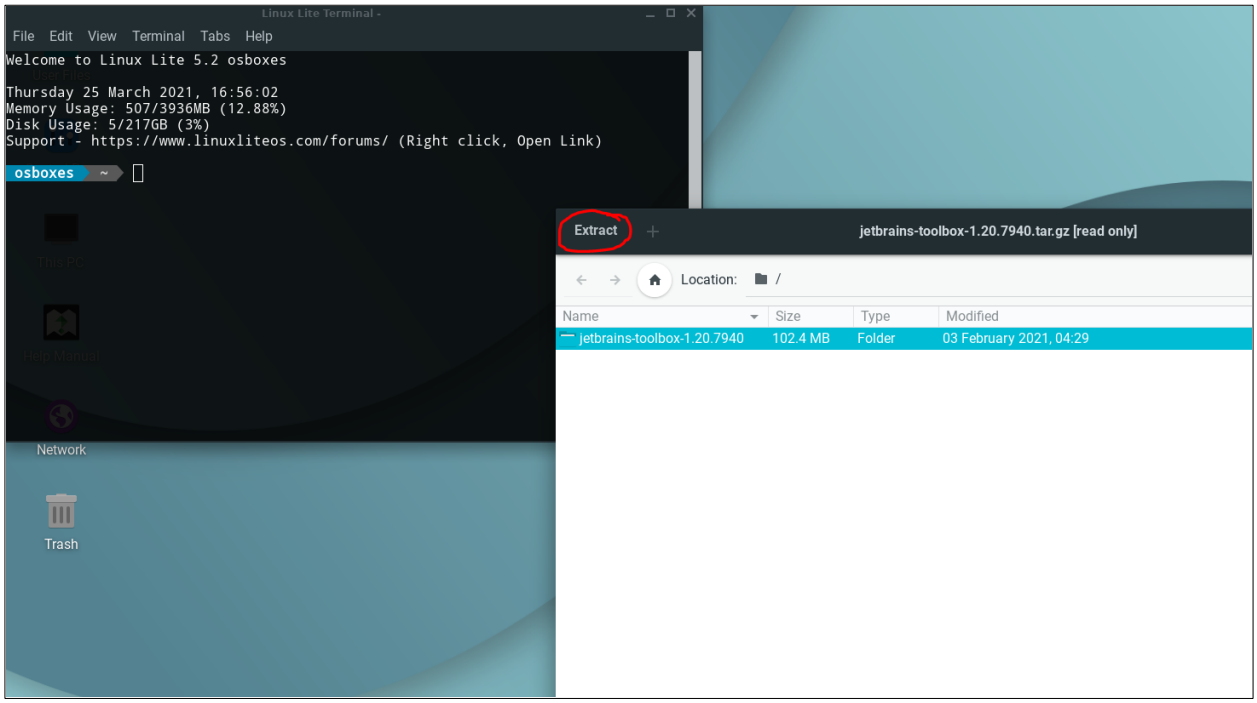
Select ‘tar.gz’ Linux.

!Depending on the version of Linux you installed, you might have an option to open the compressed file with ‘Archive Manager’ which LinuxLite has installed by default. Otherwise, you may need to just choose to ‘save the file’ and install a separate package to decompress the files. For this tutorial, we will use the Archive Manager.



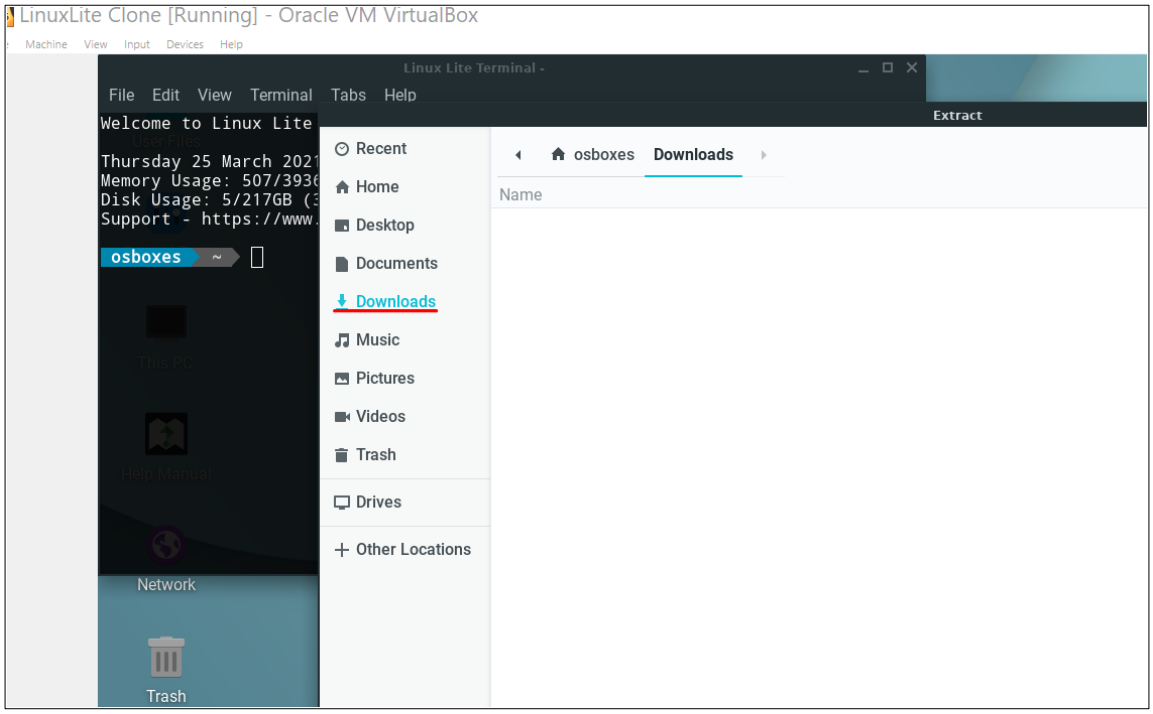
With the Archive Manager open, click on the ‘Extract’ link in the top right corner of the window.

**Time may vary in download speeds. *Your address most likely will be different.



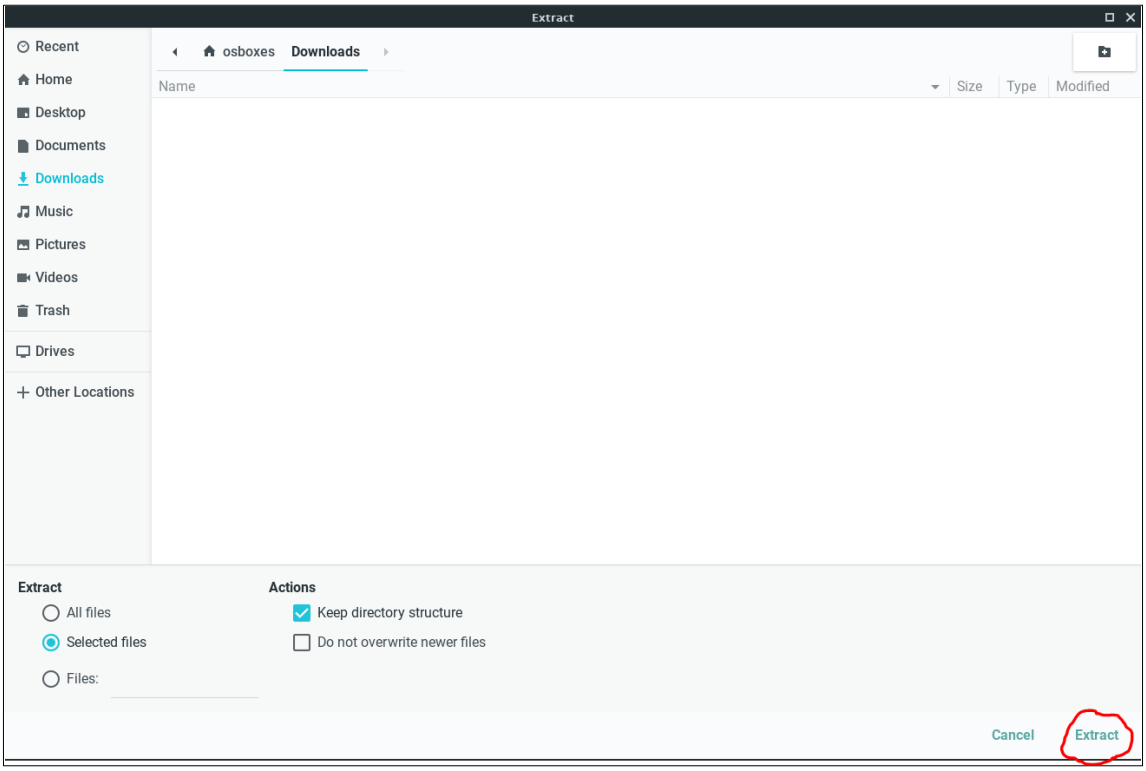
Click the path to Downloads

**Time may vary in download speeds. *Your address most likely will be different.



Now click “Extract”

**Time may vary in download speeds. *Your address most likely will be different.



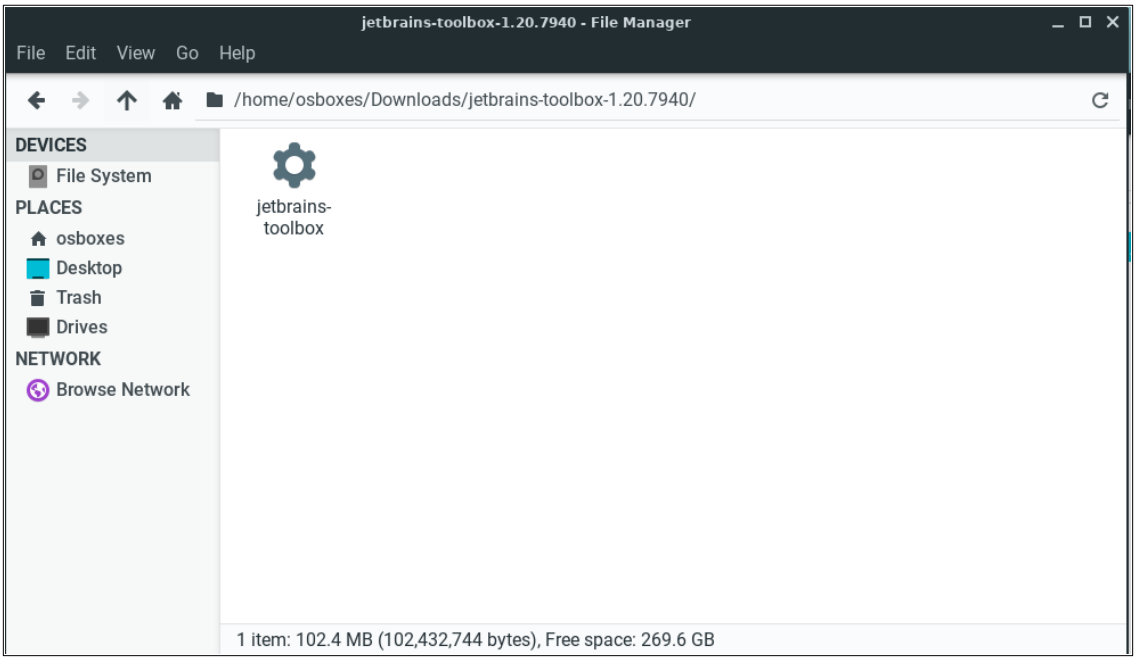
You will now see the files extracted into the Download folder.

!If you are already knowlegable of the command line, you can alternatively use this tutorial here: https://shreya-singh.medium.com/how-to-install-pycharm-in-ubuntu-16-04-ubuntu-14-04-ubuntu-18-04-linux-easiest-way-5ae19d052693

The Archive Manager completes the task and prompts you to Show the files.

Click okay and the File Manager shows a new folder called “jetbrains-toolbox-1.20.7940”. Click on that folder. Your path should look similar to the following screenshot below:

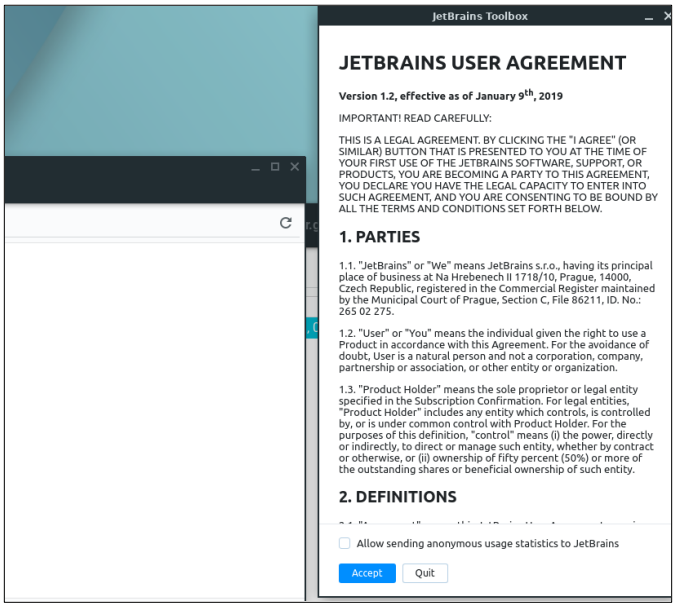
**Time may vary in download speeds. *Your address most likely will be different.



You can double click on the gear icon, or right click on it and choose execute.

!A common issue for new users of linux is whether or not the current user is authorized to execute a script. This version of osboxes doesn't have that issue and will launch the EULA from jetbrains. Other versions of linux may force you to enter a password if your account is able to elevate the execution of a script. If you have any issues with this reach out to the TA or myself with the subject line "Can't launch jetbrains-toolbox Module 7 HOP".

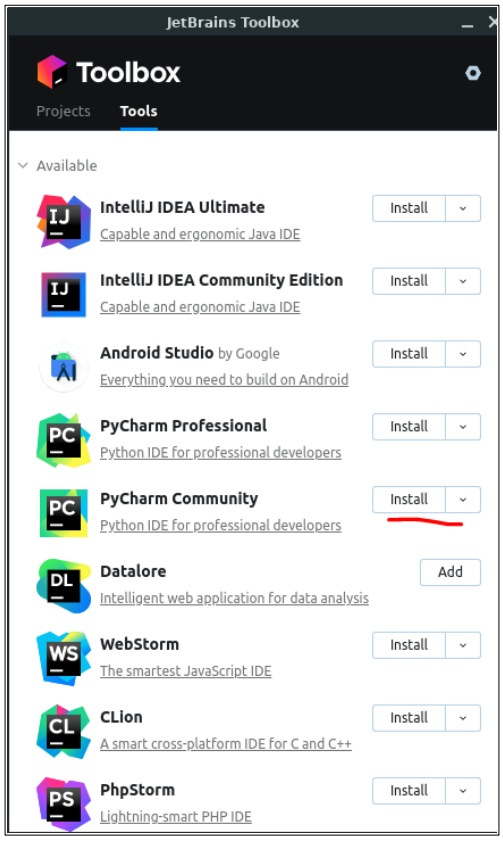
**Time may vary in download speeds. *Your address most likely will be different.



Choose whether or not to “Allow sending anonymous usage statistics to JetBrains” and click “Accept”.

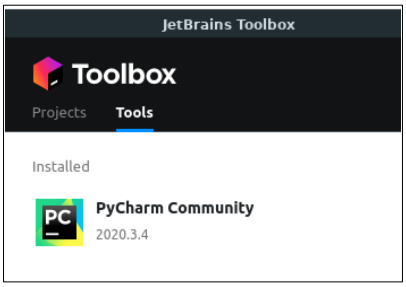
You now have a choice of Jetbrains products to choose from. The PyCharm Community is free and if you have a student account with CityU, inquire of Jetbrains if you can get a ‘student’s license’ for free. For now, the Community version is acceptable.

**Time may vary in download speeds. *Your address most likely will be different.



The installer will now download all the files and execute the installation of PyCharm**.

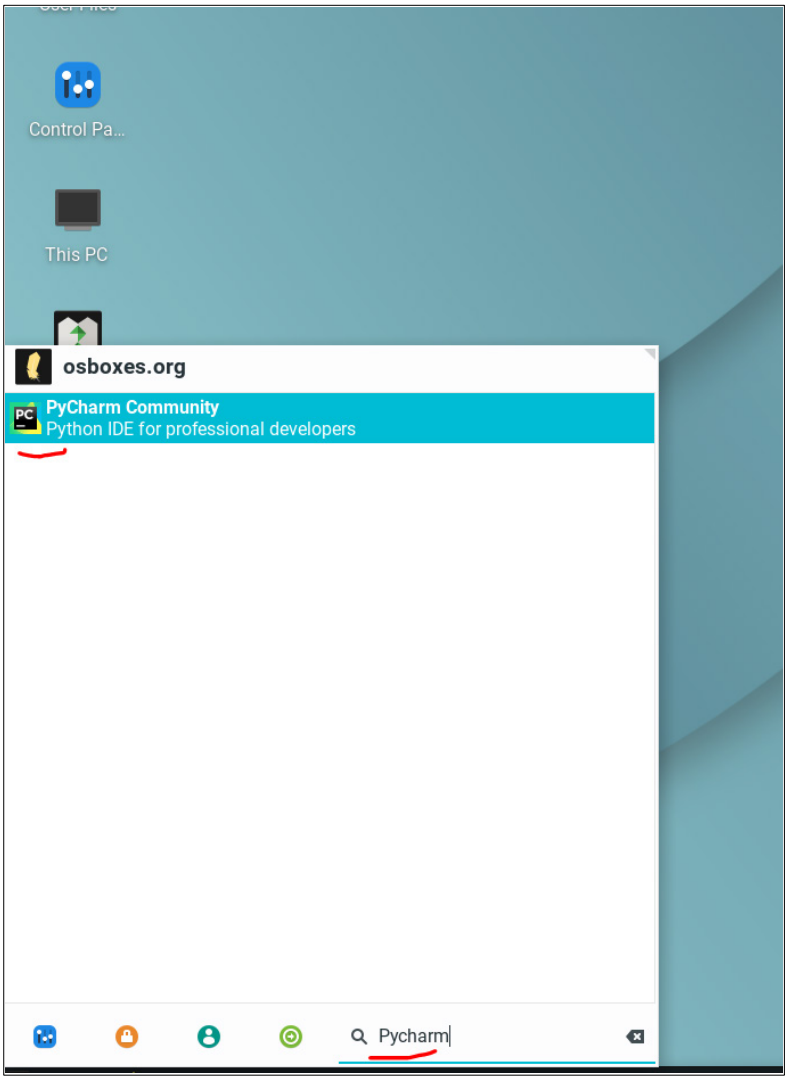
A small popup window will confirm that PyCharm is installed. The toolbox will reflect that fact:



Launch Jetbrains from the search bar in LinuxLite

**Time may vary in download speeds. *Your address most likely will be different.

At this point in the tutorial, the PyCharm IDE is successfully installed. Launching this will open your IDE to start coding in Python!



Assignment Task #2:Open Pycharm and complete the following option(s).

1.) Create a new text in Pycharm and tell me how this tutorial was. Make a screenshot and load it into HOP07 document.

2.) Create a new pycharm python project and create a new .py document. You can google a 'hello world' python script and screenshot that.

3.)For those that chose NOT to do this tutorial, screenshot what package you installed and why.

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

**Time may vary in download speeds. *Your address most likely will be different.

IS 340 – Operating Systems
HOP08 – BASH – Shell Functions
7/16/2019 Developed by Kevin Wang
6/18/2020 Reviewed by Kim Nguyen
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- This exercise assumes that the user is working with the Ubuntu 18.04 distribution. If you are working with a different Linux distribution, the set of shell commands may vary from those available in Ubuntu 18.04.
- Students will use the EC2 Ubuntu virtual machine that they created in the module 1 exercise.
- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes
Students will be able to:

- Define functions in the shell
- Define functions in the file
- Use functions in the file

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

Preparation

1) Connect to your Ubuntu instance
   Open a command prompt
   Syntax: ssh -i LOCATION_OF_YOUR_KEY ubuntu@PULIC_DNS
   Example:
   >>>ssh -i key.pem ubuntu@ec2-33-222-101-222.us-west-2.compute.amazonaws.com

2) Navigate to your name directory under the IS340-Summer-2020

>>> cd ~/ IS340-Summer-2020/YOURNAME
Note: change YOURNAME to your real name

3) Create a Module8 directory under YOURNAME directory.
Note: If this directory exists, skip this step.
>>> mkdir Module8

4) Navigate to the Module8 directory.
>>> cd Module8

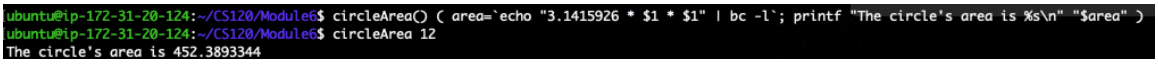
Define a function in the shell

1) The bash supports three different syntaxes for defining a function:
   - function name <compound command>
   - name() <compound command>
   - function name() <compound command>

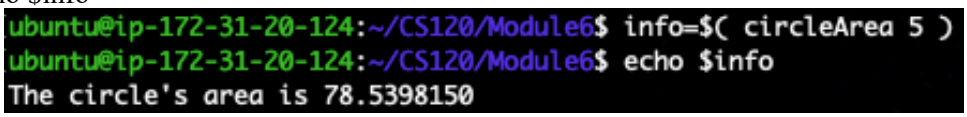
   In this module, we will use the second syntax.

2) Define a function in the shell by typing the following command:
>>> circleArea() (area=`echo "3.1415926 * $1 * $1" | bc -l`; printf "The circle's area is %s\n" "$area")
Note: the script uses the bc command to calculate float numbers. The syntax is variable=`echo "formula" | bc -l`

3) Test the script by typing the following commands:
>>> circleArea 12



4) You can save the print result to a variable for future usage. Try it by typing the following commands:
>>> info=$(circleArea 5)
>>> echo $info



Note: you can use this method to get a return value from a function

5) The function also can return an exit code (0 - 255). Test it by typing the following commands:
>>> getNumber() (return $(($RANDOM * 255 + 1)))
>>> getNumber
>>> echo $?
>>> getNumber
>>> echo $?
>>> getNumber
>>> echo $?

```
ubuntu@ip-172-31-20-124:~/CS120/Module6$ getNumber() { return $(( $RANDOM * 255 + 1 )) }
ubuntu@ip-172-31-20-124:~/CS120/Module6$ getNumber
ubuntu@ip-172-31-20-124:~/CS120/Module6$ echo $?
70
ubuntu@ip-172-31-20-124:~/CS120/Module6$ getNumber
ubuntu@ip-172-31-20-124:~/CS120/Module6$ echo $?
140
ubuntu@ip-172-31-20-124:~/CS120/Module6$ getNumber
ubuntu@ip-172-31-20-124:~/CS120/Module6$ echo $?
143
```

Note: The $? represents the last exit code which we returned from the function. You are supposed to see different numbers since our function returns a random number every time it is executed.

Define a function in the file

1) Functions defined in the shell will lose after you close the shell. It is good to write your functions in the file for the reusability

2) Create a script file by typing the following command:
>>> nano SystemSnapshot.sh

```
#! /bin/bash

snapshot() {
  filename=$(date '+%Y-%m-%d%H:%M:%S').log
  top -n 1 | tee "$filename"
}
```

Note: we define a function called snapshot to take a system snapshot by running top command and save the information to a file with the name contains date and time that snapshot was taken.

3) Hit the control + x key to quit and save the file.

4) Before we can use the function that defines in a file, we have to source in the file in the current shell, making functions available to the shell. Run the following command to source in the file:
>>> . SystemSnapshot.sh

5) Try to run the function by typing the following command:
>>> snapshot
>>> clear

We run clear to clear the screen in order to see the file content later.

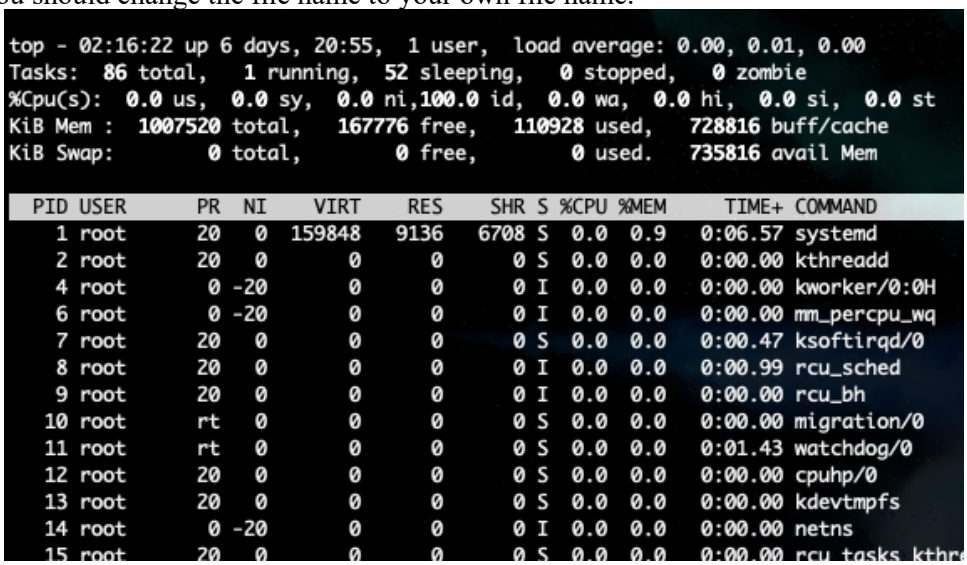
6) Run the following command to see file names in the folder:
>>> ls

```
ubuntu@ip-172-31-20-124:~/CS120/Module6$ ls
2019-07-1802:16:22.log  SystemSnapshot.sh
```

You will see a different file name because it was generated by the date and time you run the function.

7) Check the file content by typing the following command:

>>> cat 2019-07-1802:16:22.log
Note: you should change the file name to your own file name.



Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 – Operating Systems
HOP09 – BASH – Parameters and Variable
7/16/2019 Developed by Kevin Wang
6/17/2020 Reviewed by Kim Nguyen
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- This exercise assumes that the user is working with the Ubuntu 18.04 distribution. If you are working with a different Linux distribution, the set of shell commands may vary from those available in Ubuntu 18.04.
- Students will use the EC2 Ubuntu virtual machine that they created in the module 1 exercise.
- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

- Understand the scope of variables
- Use the parameter expansion
- Use arrays

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

Preparation

1) Connect to your Ubuntu instance

Open a command prompt

Syntax: ssh -i LOCATION_OF_YOUR_KEY ubuntu@PUBLIC_DNS

Example:

>>>ssh -i key.pem ubuntu@ec2-33-222-101-222.us-west-2.compute.amazonaws.com

2) git pull https://github.com/cityuseattle/IS340-Fall-2020-Assignment.git (to get the most updated content from source repository). If you are prompted to type a message, you can skip this by typing :wq + Enter **!!! NOTE: FOR EACH WEEK, YOU NEED TO DO THIS STEP BEFORE YOU START CODING)!!!**
3) Change directory to the corresponding folder of each week. For example: Your work for module 1 should be stored under Module 1 folder; your work for module 2 should be stored under Module 2, and so on:

- cd "Module 7"

4) Now, follow the instructions provided in each folder to complete your Hands-on Practice

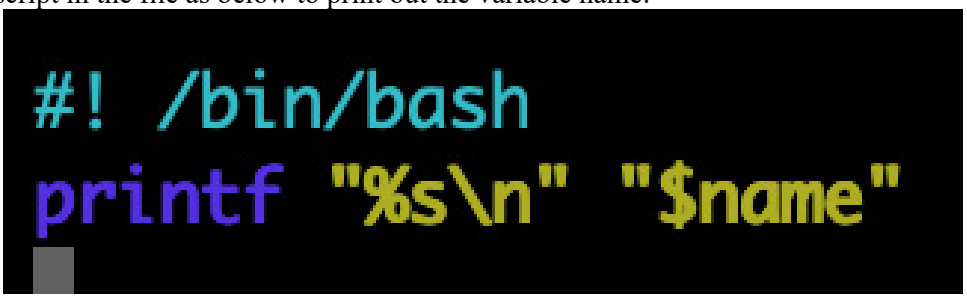
Variable Scope

1) The variable is defined in the shell will be just visible in the shell. No script that is called by the shell can see it. Test this by following the instructions below:

   Type the following command to define a variable in the shell:
   >>> name=Kevin

   Type the following command to create a script file:
   >>> nano NameTest.sh

   Type the script in the file as below to print out the variable name:



2) Hit the control + x key to quit and save the file

3) Test the script by typing the following commands:
   >>> bash NameTest.sh
   Note: you are supposed to see an empty output since the name variable is defined in the shell instead of in the script file (The script knows nothing about it).

4) Export the variable to the environment by typing the following commands:
   >>> export name=Kevin
   Note: we export the name variable to the environment.

5) Try to run the script again:
   >>> bash NameTest.sh

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ export name=Kevin
ubuntu@ip-172-31-20-124:~/CS120/Module5$ bash NameTest.sh
Kevin
```

Now our script can read the variable name we defined in the shell.

6) Use unset command to remove a variable from the environment:
>>> unset name
>>> bash NameTest.sh
You will see the empty output again

7) The variables are defined in the subshell will not be visible to the shell that called it. This is a common problem when we use the pipe functionality. Type the following command to create a file to test it:
>>> nano SubshellTest.sh

8) Type the following script in the file:

```
#! /bin/bash
sum=0
printf "Add sum to these numbers:\n"
printf "%s\n" ${RANDOM}{,,,} |
while read num
do
  printf "The current number is %d\n" "$num"
  ((sum=sum+num))
  printf "The current sum number is %d\n" "$sum"
done

printf "Now the loop is finished.\n\nThe final sum is %d\n" "$sum"
```

Note: we generate several random numbers and use the pipe to pass them to a while loop. The while loop adds them together and put the total value to the sum variable.

9) Hit the control + x key to quit and save

10) Type the following command to run the script:
>>> bash SubshellTest.sh

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ bash SubshellTest.sh
Add sum to these numbers:
The current number is 17811
The current sum number is 17811
The current number is 13875
The current sum number is 31686
The current number is 30364
The current sum number is 62050
The current number is 3513
The current sum number is 65563
Now the loop is finished.

The final sum is 0
```

As you can see from the output, the sum variable is visible inside of the while loop but not visible after the while loop. This is because we are using the pipe operation | to run the while loop, which causes it runs in a subshell and the variable in the subshell is not visible in the caller environment.

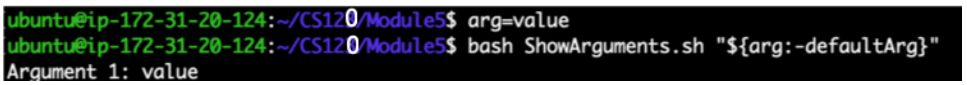
Parameter Expansion

1) Copy the ShowArguments.sh file we made in the module7 by typing the follow command:
   >>> cp ../Module7/ShowArguments.sh ./

2) Type the following command to test how to give a default value to a parameter:
   >>> arg=
   >>> bash ShowArguments.sh "${arg:-defaultArg}"

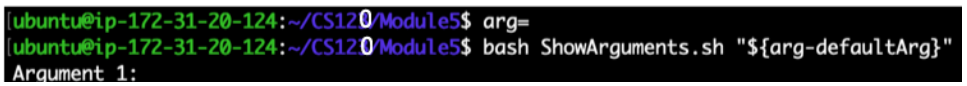
   Note: we set an empty arg as the argument and call the script with a default value “defaultArg”. So, the script will take the defaultArg as the parameter value.

   >>> arg=value
   >>> bash ShowArguments.sh "${arg:-defaultArg}"



   This time we defined a value for arg variable. So, when we run the script, it will use that value instead of the default value.

3) If we omit the colon, the script will just check whether the variable is unset. Test it by typing the following commands:
   >>> arg=
   >>> bash ShowArguments.sh "${arg-defaultArg}"



   Even the arg is no value, the default value will not be applied since the script knows it is not unset.

4) ${var:=default} and ${var=default} can define a default value for a variable. Test it by typing the follow command:
   >>> unset arg
   >>> echo $((1+${arg:=100}))

5) ${var:?message} and ${var?message}will show the message when the variable is empty or unset. Test it by typing the follow command:
   >>> arg=
   >>> echo $((1+${arg:?EmptyVariable}))

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ arg=
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo $((1+${arg:?EmptyVariable}))
-bash: arg: EmptyVariable
```

6) ${#var} can show the length of a variable. Test it by typing the follow command:
>>> var=text
>>> echo ${#var}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ var=text
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${#var}
4
```

7) ${var%PATTERN} will remove the shortest match from the end. Test it by typing the follow command:
>>> var=testtexttext
>>> echo ${var%text*}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ var=testtexttext
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${var%text*}
testtext
```

8) ${var%%PATTERN} will remove the longest match from the end. Test it by typing the following command:
>>> echo ${var%%text*}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${var%%text*}
test
```

9) ${var#PATTERN} and ${var##PATTERN} will remove the shortest and longest match from the beginning. Test them by typing the follow command:
>>> var=testtesttext
>>> echo ${var#*test}
>>> echo ${var##*test}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ var=testtesttext
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${var#*test}
testtext
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${var##*test}
text
```

10) ${var//PATTERN/STRING} can help to replace the match pattern with the given string. Test it by typing the follow command:
>>> var=teabct
>>> echo ${var/abc/s}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ var=teabct
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${var/abc/s}
test
```

11) ${var:OFFSET:LENGTH} can return a substring of a variable. Test it by typing the following command:
>>> var=testtext
>>> echo ${var:4}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ var=testtext
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${var:4}
text
```

Note: the length is optional. If a length parameter is not supplied, the command will return all string from the offset position.

Using arrays

1) Create an array by typing the follow command:
>>> unset name
>>> name[0]=Kevin
>>> name[1]=Arthur
>>> echo ${name[0]}
>>> echo ${name[1]}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ unset name
ubuntu@ip-172-31-20-124:~/CS120/Module5$ name[0]=Kevin
ubuntu@ip-172-31-20-124:~/CS120/Module5$ name[1]=Arthur
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${name[0]}
Kevin
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${name[1]}
Arthur
```

2) You also can set an array with all value in one command:
>>> unset name
>>> name=(Kevin Evan Arthur)
>>> printf "%s\n" "${name[@]}"

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ unset name
ubuntu@ip-172-31-20-124:~/CS120/Module5$ name=( Kevin Evan Arthur )
ubuntu@ip-172-31-20-124:~/CS120/Module5$ printf "%s\n" "${name[@]}"
Kevin
Evan
Arthur
```

3) Associative Arrays allows developers to use strings as index. Test it by typing the follow commands:
>>> declare -A strArr
>>> strArr[kevin]="Kevin's address"
>>> strArr[evan]="Evan's address"
>>> echo ${strArr[kevin]}
>>> echo ${strArr[evan]}

```
ubuntu@ip-172-31-20-124:~/CS120/Module5$ declare -A strArr
ubuntu@ip-172-31-20-124:~/CS120/Module5$ strArr[kevin]="Kevin's address"
ubuntu@ip-172-31-20-124:~/CS120/Module5$ strArr[evan]="Evan's address"
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${strArr[kevin]}
Kevin's address
ubuntu@ip-172-31-20-124:~/CS120/Module5$ echo ${strArr[evan]}
Evan's address
```

Note: the associative arrays that use string as indexes must be declare before use.

Submit your Work to Brightspace

Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.

IS 340 – Operating Systems
HOP10 – BASH – String Manipulation
7/18/2019 Developed by Kevin Wang
7/20/2020 Reviewed by Kim Nguyen
02/14/2022 Reviewed by Ken Ling

School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- This exercise assumes that the user is working with the Ubuntu 18.04 distribution. If you are working with a different Linux distribution, the set of shell commands may vary from those available in Ubuntu 18.04.
- Students will use the EC2 Ubuntu virtual machine that they created in the module 1 exercise.
- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below and experiment in the Ubuntu console and try to solve the problem yourself.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

- Concatenate strings
- Process character by character and reverse a string
- Convert cases
- Insert characters into a string
- Trim unwanted characters

Resources

- Linux command line: **bash + utilities**
  https://ss64.com/bash/

- Nano/Basics Guide
  https://wiki.gentoo.org/wiki/Nano/Basics_Guide

Preparation

1) Connect to your Ubuntu instance
   Open a command prompt
   Syntax: ssh -i LOCATION_OF_YOUR_KEY ubuntu@PULIC_DNS
   Example:
   >>>ssh -i key.pem ubuntu@ec2-33-222-101-222.us-west-2.compute.amazonaws.com

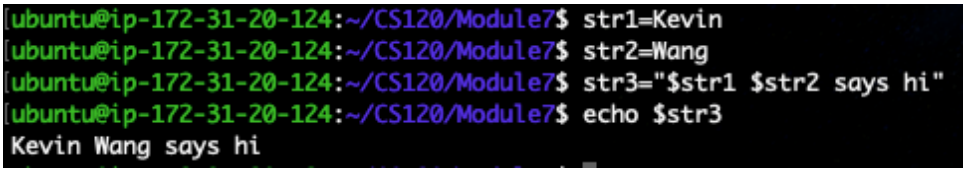
2) Navigate to your name directory under the IS340-Summer-2020
>>> cd ~/ IS340-Summer-2020/YOURNAME
Note: change YOURNAME to your real name

3) Create a Module9 directory under YOURNAME directory.
Note: If this directory exists, skip this step.
>>> mkdir Module9

4) Navigate to the Module9 directory.
>>> cd Module9

Concatenate strings

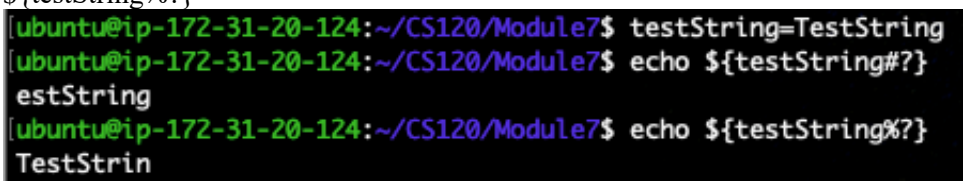
1) Concatenating strings is very easy in the Bash. Try it by typing follow commands:
>>> str1=Kevin
>>> str2=Wang
>>> str3="$str1 $str2 says hi"
>>> echo $str3



2) After Bash 3.1, we can use += operator to concatenate strings. Test it by typing the following command:
>>> str3+=" there."
>>> echo $str3
Note: directly copy commands will not work since the Word uses different double quote marks. Please type the command by yourself.

Process character by character and reverse a string

1) Test the following commands to see how bash script can get whole string without the first or last character:
>>> testString=TestString
>>> echo ${testString#?}
>>> echo ${testString%?}



Note: #? means marking one character from beginning and %? means marking one character from the end. You can use #?? to mark two characters.

2) So, we can use the follow combined commands to get the first or last character:
>>> echo ${testString%${testString#?}}
>>> echo ${testString#${testString%?}}

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo ${testString%${testString#?}}
T
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo ${testString#${testString%?}}
g
```

3) Try to use the technic we learned above to make a function that helps to reverse a string. Type the following command to create a script file:
>>> nano Reverse.sh

4) Type the script into the file as below:

```
#! /bin/bash

reverse() {
  str=$1
  reversedStr=
  while [ -n "$str" ]
  do
    temp=${str%?}
    reversedStr=$reversedStr${str#$temp}
    str=$temp
  done
  printf "The reversed string is: %s\n" "$reversedStr"
}
```

5) Hit the control + x to quit and save the file
6) Type the following command to source the file:
>>> . Reverse.sh

7) Type the following command to test the reverse function:
>>> reverse abcde

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ . Reverse.sh
ubuntu@ip-172-31-20-124:~/CS120/Module7$ reverse abcde
The reversed string is: edcba
```

Case conversion

1) The tr command can help to make a character map. Test it by typing the following commands:
>>> echo abc | tr abc ABC
>>> echo abc | tr c C
>>> echo abc | tr abc efg

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo abc | tr abc ABC
ABC
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo abc | tr c C
abC
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo abc | tr abc efg
efg
```

Note: as you can see the tr command is offering a map that can map a character to any other character.

2) Try this command to map a group of characters:
>>> echo “This is a full sentence.” | tr ‘a-z’ ‘A-Z’

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo "This is a full sentence." | tr 'a-z' 'A-Z'
THIS IS A FULL SENTENCE.
```

Note: please do not directly copy the command.

3) Sometimes, we want to compare two strings that have the same characters with different cases. We can convert both strings to uppercase and compare them. Try it by typing the following commands:
>>> str1=abc
>>> str2=aBc
>>> [[$str1 == $str2]] && echo equal || echo "Not equal"

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ str1=abc
ubuntu@ip-172-31-20-124:~/CS120/Module7$ str2=aBc
ubuntu@ip-172-31-20-124:~/CS120/Module7$ [[ $str1 == $str2 ]] && echo equal || echo "Not equal"
Not equal
```

The result shows they are not equal since we have different cases for the letter b.
Use the following commands to overcome this:
>>> [[${str1^^} == ${str2^^}]] && echo equal || echo "Not equal"

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ [[ ${str1^^} == ${str2^^} ]] && echo equal || echo "Not equal"
equal
```

Insert characters in a string

1) In order to perform the insertion, we have to separate the original string to the left and right part. Try this by typing the following commands:
>>> str1=abcd
>>> echo ${str1:0:2}
>>> echo ${str1:2}

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ str1=abcd
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo ${str1:0:2}
ab
ubuntu@ip-172-31-20-124:~/CS120/Module7$ echo ${str1:2}
cd
```

2) Now we can use this approach to make an insert function. Create a script file by typing the following command:
>>> nano Insert.sh

3) Type the following script in the file:

```
#! /bin/bash
insert() {
  str=$1
  left=${str:0:$(( $3 - 1 ))}
  right=${str:$(( $3 - 1 ))}
  printf "The final string is: %s\n" "$left$2$right"
}
```

4) Hit the control + x key to quit and save the file:

5) Test the following commands to test the new function:
>>> . Insert.sh
>>> insert Hllo e 2

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ . Insert.sh
ubuntu@ip-172-31-20-124:~/CS120/Module7$ insert Hllo e 2
The final string is: Hello
```

Trim unwanted characters

1) Type the following command to create a trim script:
>>> nano Trim.sh

2) Type the following script in the file:

```
#! /bin/bash
trim() {
  str=$1
  left=${str%%[!$2]*}
  str=${str#"$left"}
  right=${str##*[!$2]}
  str=${str%"$right"}
  printf "The final string is: %s\n" "$str"
}
```

3) Hit the control + x key to quit and save the file.

4) Test the script by typing follow commands:
>>> . Trim.sh
>>> trim 0000213.43000 0

```
ubuntu@ip-172-31-20-124:~/CS120/Module7$ . Trim.sh
ubuntu@ip-172-31-20-124:~/CS120/Module7$ trim 0000213.43000 0
The final string is: 213.43
```

Push your work to GitHub

Run the following commands to push your work to the GitHub repository:
>>> git add .
>>> git commit -m “Submission for Module 10”
>>> git push origin YOUR_BRANCH_NAME

Note: you should change the YOUR_BRANCH_NAME to your own branch name. It should be firstname-lastname (e.g. maria-gracia).
If you cannot remember, run the command “git status” to check.